Language-based Performance Prediction for Distributed and Mobile Systems

Corrado Priami

Dipartimento di Informatica e TLC, Università di Trento, Via Sommarive 14, I-38050 Povo, Italy
E-mail: priami@dit.unitn.it

We present a framework for performance prediction of distributed and mobile systems. We rely on process calculi and their structural operational semantics. The dynamic behaviour is described through transition systems whose transitions are labelled by encodings of their proofs that we then map into stochastic processes. We enhance related works by allowing general continuous distributions resorting to a notion of enabling between transitions. We also discuss how the number of resources available affects the overall model. Finally, we introduce a notion of bisimulation that takes stochastic information into account and prove it to be a congruence. When only exponential distributions are of interest our equivalence induces a lumpable partition on the underlying Markov process.

Key Words: operational semantics; proved transition system; stochastic process algebra; general distributions; performance prediction.

1. INTRODUCTION

The wide dissemination of Internet sites and the emerging paradigm of global computations based on the mobility of both code and computations make quantitative analysis of specification as important as qualitative analysis. The need for integration of qualitative and quantitative analyses in developing complex systems since the early stages of projects has been well accepted. Full integration is presented as a challenge for the future of computer science in [14].

An attempt at integration is given by Markovian and stochastic process algebras [4, 7–9, 11, 15, 16, 24, 27, 30, 31, 44, 45]. Process algebras are foundational calculi used to describe the concurrent and distributed structure of systems. They are made up of a few operators such as: (i) $a.-$ that which describes sequential composition of actions, (ii) $-\mid-$ that which is the parallel composition of processes, (iii) $-+-$ that which denotes a nondeterministic choice. We can view process algebras from different levels of abstraction. A common interpretation is seeing these calculi as specificatio languages that must be refine towards a real code. The theory of behavioral analysis developed for process algebras (see [35]) postulates that refine descriptions are still expressed in the same calculus but through different programs. Then, some relations (usually a bisimulation) are established between the two descriptions of the system to ensure that an implementation behaves according to its specification We are proposing here a different use of process algebras. We interpret these calculi as an intermediate language into which programs written in real languages can be translated. For instance, languages like *Facile* [23], *CML* [39], and *Pict* [42] can be easily mapped to process algebras being define as directly extending them. We will come back to this interpretation of process algebras when we describe our approach to quantitative analysis. For the time being, we assume the classical interpretation as specificatio languages to survey stochastic extensions of these calculi.

Stochastic process algebras enrich the actions of classical calculi with continuous probabilistic distributions, yielding prefi es such as (a, F). The distributions F are assumed to be exponential in almost all the proposals. Unfortunately, many phenomena that arise in practice are naturally described by nonexponential distributions. Consider, for instance, the routing of a message on a network. Even if the transfer of the message between two adjacent nodes exposes an exponential delay, the overall routing may not. Furthermore, the industrial environment calls for general distributions, especially in the areas of workfl w models, robot systems, and ATM networks [29]. Another limitation of stochastic process algebras concerns the modificatio of the syntax of the language by inserting probabilistic distributions within prefi es. Therefore, the designer must specify the intended behavior of a system once he or she has in mind all the features of the architecture on which the specifi cation will be implemented. Otherwise, there is little hope of associating suitable distributions with prefi es. For instance, the usage of a mobile agent or of a remote procedure call for an interaction

0890-5401/02 $35.00

must be decided when designing a system. But the fewer details needed at specification time, the better.

Our idea is to retain the advantages of stochastic process algebras without modifying the syntax of calculi that remain independent of the architectural aspects of implementations. We only change the abstraction level at which process algebras are interpreted. The association of probabilistic distributions with actions is then a matter of the compiler or the interpreter of the language that necessarily has all the relevant information about the target architecture. In this paper we concentrate on how the compiler associates quantitative information with actions. The translation of real languages to process algebras is out of the scope of this work (some examples can be found in the literature [23]). We start with a description of the system given via a process algebra allowing for code mobility. For instance, the π-calculus [36], *Plain-CHOCS* [52], the *join calculus* [22], Dπ [50], and the *ambient calculus* [12] have recently been proposed as an evolution of classical process calculi and they handle mobility naturally. Here we use the π-calculus that permits port names to be transmitted from one process to another in communications. Consequently, the interconnection structure of a network is no longer static, but can vary dynamically when processes communicate. This simple conceptual extension is sufficient to describe the mobility of processes [51].

The quantitative parameters that we associate with actions are determined on semantic grounds along the lines of [10, 40, 41]. Here we use structural operational semantics (hereafter abbreviated as SOS) [43], the usual way of assigning meaning to process algebras. The advantages of SOS are twofold. First, we obtain a description of the abstract machine that executes process algebra programs. Thus we have a precise description of the low-level operations needed to execute the actions of the program. Second, the SOS definitions are logical in style, mathematically simple, and appealing. An example is the rule for the parallel composition of processes:

$$\frac{P \xrightarrow{\theta} P'}{P|Q \xrightarrow{\theta} P'|Q}.$$

The interpretation of this inference rule is that whenever the premise occurs (that is, satisfied), the conclusion occurs as well. Therefore, if P can move to P' via a θ action, the parallel composition $P|Q$ can perform the same action and becomes the configuration $P'|Q$.

We can now see the derivation tree of a transition (i.e., the set of inference rules applied in the derivation) as an encoding of the low level operations of the abstract machine needed to perform the action corresponding to the transition. We thus enrich transition labels with a linearization of their deduction trees. Look again at the rule for parallel composition:

$$\frac{P \xrightarrow{\theta} P'}{P|Q \xrightarrow{||_0\theta} P'|Q}.$$

In the conclusion label we recorded the application of the rule via the tag $||_0$, where 0 means that the left component is moving. The new operational semantics is called *proved* [6, 18]. A version of this semantics is available for the π-calculus [20] and for the real programming languages Facile [21] and Esterel [34].

We implement a \$ function that takes as arguments the new transition labels and provides us with a parameter expressing the execution cost (duration) of the transition. The \$ function encodes the cost of executing the run-time support routines of the language to perform the action corresponding to a transition. Unfortunately, this is only possible when the history of the system does not influence the execution of the current transition. In other words, the system must have a memoryless property. Consequently, a function such as the one above is only suitable when durations of transitions are exponentially distributed. But we have already pointed out the limitation of exponential distributions. Therefore we need to know the enabling relation between the transitions of the system in order to handle general distributions. This allows us to keep track of the time spent by an action in the states in which it was enabled but not selected to fire. Once again the deduction trees (or better their linearizations) help

us in the def nition of enabling. In fact, we only need to establish a pref x relation on the strings of tags in the label of two transitions to see whether one of the two enables the other [20]. The intuitive idea is that two actions sequentially nested into a chain of pref xes are derived by using the same initial set of SOS rules. We then use this information together with the $ function to compute the f ring distributions of transitions.

The distribution of a θ transition must be inf uenced by those of all the transitions f red from the states where θ was enabled. This was called *enabling memory discipline* in [1]. The author proposed in [46] a stochastic extension of the π-calculus with general distributions, but in that case the syntax was enriched by putting distributions within pref xes.

We then introduce an enhanced version of the enabling memory discipline by using a different notion of timer associated with a transition. In classical stochastic process algebras the duration of a transition is computed by calculating the time necessary to f nish its work *when scheduled* on a CPU. Transitions do not consume time when the processes which perform them are not active. Therefore performance measures are ideal: any process always has a free processor for it. On the other hand, the measured performance of a system is in terms of CPU time consumption. This is a good index for comparing the performance of two systems, but it is far from being signif cant as far as the response time is concerned. In fact, most of the user's waiting time is due to scheduling activities and idle times. To better express response time, we assume that the timer of a transition starts when it is enabled the f rst time and stops when the corresponding activity is performed, counting idle times as well. Under this interpretation of transition timers, the stochastic semantics can no longer assume that any process has its own processor. In fact, the number of available processors greatly inf uences the idle times and hence the response time of systems. According to the above discussion, we recover in the stochastic f eld the classical interpretation that interleaving descriptions actually correspond to uniprocessor architectures. Therefore, we modify the semantics of the π-calculus to take into account the number of available processors.

Two concurrent transitions can be executed on different processors and hence their durations are independent. If we consider the transition system representation of the two concurrent transitions, we have two computations starting from the same state and leading to the same target that differ only in the order of transition executions. This representation is adequate for a uniprocessor architecture where only one process is running at a time. This interpretation leads to the def nition of enabling memory discipline as given in [46] and described above. If two processors are available, the two transitions are executed on different machines and therefore their durations must be independent. To model the number of processors available within the semantic def nition of processes, we resort to higher dimension transition systems brief y outlined in [20]. Essentially, transitions can carry over as many labels as the number n of processors. We include in the transition systems the diagonal of the concurrency diamond of dimension n. In this case, we only need to extend the def nition of concurrency to higher dimension transitions and then apply the def nition of enabling memory discipline adapted to multilabelled transitions (see also [47]).

A major problem in using general distributions to evaluate systems is their diff cult tractability. In fact, it is diff cult to f nd closed form solutions of stochastic processes and often extensive simulation is required. To improve feasibility, we introduce a class of transition systems (that of maximal parallelism) for which the computation of performance measures still using general distributions is easy. Unfortunately, we cannot derive *exact* measures, but only upper bounds to system response time.

To ease the integration of qualitative and quantitative analyses of systems, we def ne a bisimulation-based equivalence to compare program performance. It extends the late bisimulation for the π-calculus [37] in the style of [28, 30, 33]. We keep track of the enabling relation between transitions when comparing two systems to get a congruence result for the full π-calculus without matching similarly to [5]. Since equivalences are often used to implement model minimization to speed up quantitative analysis, we prove that our bisimulation induces a lumpable partition on the underlying Markov process when distributions are restricted to exponential ones.

This article is organized as follows: in the next section we introduce the basic notions of the π-calculus. Section 3 def nes its stochastic semantics. Enabling and concurrency are def ned in Section 4. They are then used to compute the distributions of transitions and eventually the stochastic process associated with a transition system. Section 5 ref nes the enabling memory discipline introduced in the previous section to cope with the amount of resources available. The case of unbound resources is

discussed as well. Section 6 introduces a notion of performance bisimulation to cope with the stochastic properties of systems. Section 7 instantiates our framework to exponential distributions. Finally, we discuss related work in Secttion 8.

2. THE π-CALCULUS

In this section we brief y review the π-calculus [36], a model of concurrent communicating processes based on the notion of *naming*.

DEFINITION 2.1. $\mathcal{N}$ is a countable inf nite set of *names* ranged over by $a, b, \ldots, x, y, \ldots$ and $\mathcal{S} = \{\tau_0, \tau_1, \tau_2, \ldots\}$ is a countable inf nite set of *invisible actions* ranged over by τ_i, with $\mathcal{N} \cap \mathcal{S} = \emptyset$. We also assume a set of *agent identifiers*, each with an arity, ranged over by $A, A_1, \ldots$. *Processes* in $\mathcal{P}$, ranged over by $P, Q, R, \ldots$ are def ned as

$$P ::= \mathbf{0} \mid X \mid \pi.P \mid (\nu x)P \mid [x = y]P \mid P|P \mid P + P \mid A(y_1, \ldots, y_n),$$

where π may be either $x(y)$ for *input* or $\bar{x}y$ for *output* (where x is the *subject* and y is the *object*) or τ_i for *silent* moves. The order of *precedence* among the operators is the order (from left to right) listed above. Hereafter, the trailing $\mathbf{0}$ will be omitted.

In the above def nition we used a set of silent moves to distinguish their different durations. We sometimes write $(\nu x, y)P$ for $(\nu x)(\nu y)P$. Each agent identif er A has a unique def ning equation in the form $A(\tilde{y}) = P$ (hereafter, $\tilde{y}$ denotes $y_1, \ldots, y_n$), where the y_i are all distinct and are the only free names in P.

The operational semantics of the π-calculus is def ned in the *SOS* style. We use μ as a metavariable for transition labels. We introduce set $\mathcal{A}$ of visible actions ranged over by α (i.e., $x(y)$ for input, $\bar{x}y$ for free output, and $\bar{x}y$ for bound output[1]). Note that the transition labels differ from pref xes π because of the presence of bound outputs.

We recall the notion of free names $fn(\mu)$, bound names $bn(\mu)$, and names $n(\mu) = fn(\mu) \cup bn(\mu)$ of a label μ; only the bound names are the objects of input and of the bound output. Functions fn, bn, and n are extended to processes by inducing on their syntax and considering input pref xes and ν operators as binders. We def ne the *structural congruence* $\equiv$ on processes as the least congruence that satisf es the following clauses:

- $P \equiv Q$ if P and Q differ only in the choice of bound names (α-equivalent),
- $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$.

We def ne our *enhanced labels* in the style of [6, 17, 20]. A transition label records the inference rules used during its deduction, besides the action itself. We call *proof term* the encoding of the proof in an enhanced label. Finally, we introduce an ℓ function that takes an enhanced label to the corresponding standard action label.

DEFINITION 2.2. If $\mathcal{L} = \{\|_0, \|_1\}$ with $\chi \in \mathcal{L}^*$, $\mathcal{O} = \{+_0, +_1, =_m, (\nu x), (\tilde{y})\}$ with $o \in \mathcal{O} \cup \mathcal{L}$, and $\vartheta \in (\mathcal{L} \cup \mathcal{O})^*$, then the set Θ of *enhanced labels* (with metavariable θ) is def ned by the syntax

$$\theta ::= \vartheta\alpha \mid \vartheta\tau_i \mid \vartheta\langle \|_0\vartheta_0\alpha_0, \|_1\vartheta_1\alpha_1\rangle,$$

where $\alpha_0 = x(y)$ iff α_1 is either $\bar{x}y$ or $\bar{x}y$, and vice versa.

Function ℓ is def ned as $\ell(\vartheta\alpha) = \alpha$, $\ell(\vartheta\tau_i) = \ell(\vartheta\langle \|_0\vartheta_0\alpha_0, \|_1\vartheta_1\alpha_1\rangle) = \tau$.

A $+_0$ ($+_1$) tag means that a nondeterministic choice has been made in favour of the left (right) component. Similarly, a $\|_0$ ($\|_1$) tag records that the left (right) component of a parallel composition

[1] The effect of a bound output is vanishing a ν operator. Consider for instance the transition $Q = (\nu x)\bar{y}x.P \xrightarrow{\bar{y}(x)} P$. The intuition behind this operation is to make the private name x of Q available to the external environment. In fact, operator ν can be interpreted as a delimiter of an environment, while the bound output is an open of that environment. This is why the bound output is sometimes referred to as scope extrusion in the literature.

TABLE I

Proved Transition System for the π-Calculus

$$Act\colon \pi.P \xrightarrow{\pi} P \qquad Ide\colon \frac{P\{\tilde{y}/\tilde{x}\} \xrightarrow{\theta} P'}{Q(\tilde{y}) \xrightarrow{(\tilde{y})\theta} P'},\ Q(\tilde{x}) = P$$

$$Par_0\colon \frac{P \xrightarrow{\theta} P'}{P|Q \xrightarrow{||_0\theta} P'|Q},\ bn(\ell(\theta)) \cap fn(Q) = \emptyset \qquad Sum_0\colon \frac{P \xrightarrow{\theta} P'}{P + Q \xrightarrow{+_0\theta} P'}$$

$$Par_1\colon \frac{P \xrightarrow{\theta} P'}{Q|P \xrightarrow{||_1\theta} Q|P'},\ bn(\ell(\theta)) \cap fn(Q) = \emptyset \qquad Sum_1\colon \frac{P \xrightarrow{\theta} P'}{Q + P \xrightarrow{+_1\theta} P'}$$

$$Res\colon \frac{P \xrightarrow{\theta} P'}{(\nu x)P \xrightarrow{(\nu x)\theta} (\nu x)P'},\ x \notin n(\ell(\theta)) \qquad Open\colon \frac{P \xrightarrow{\vartheta \bar{x} y} P'}{(\nu y)P \xrightarrow{\vartheta \bar{x} y} P'},\ x \neq y$$

$$Com_0\colon \frac{P \xrightarrow{\vartheta \bar{x} y} P',\ Q \xrightarrow{\vartheta' x(w)} Q'}{P|Q \xrightarrow{\langle ||_0 \vartheta \bar{x} y, ||_1 \vartheta' x(w)\rangle} P'|Q'\{y/w\}}$$

$$Close_0\colon \frac{P \xrightarrow{\vartheta \bar{x} y} P',\ Q \xrightarrow{\vartheta' x(w)} Q'}{P|Q \xrightarrow{\langle ||_0 \vartheta \bar{x} y, ||_1 \vartheta' x(w)\rangle} (\nu y)(P'|Q'\{y/w\})},\ y \notin fn(Q)$$

is moving. Restriction is reported on the labels to record that a filter has been passed. We record the resolution of a matching through $=_m$ tag, where m is the size of the data to be compared. Communications are labelled by a pair instead of a τ to show the components which interacted (and proof of the relevant transitions). We also record in the labels the actual parameters $\tilde{y}$ of a definition because their number and size affect the instantiation cost.

Our transition system for the π-calculus is shown in Table 1. A *variant* of $P \xrightarrow{\mu} Q$ is a transition which only differs in that P and Q have been replaced by structurally congruent processes, and μ has been α-converted, where a name bound in μ includes Q in its scope [37]. For instance, $x(y).\bar{y}z.P \xrightarrow{x(y)} \bar{y}z.P\{y/w\}$ is a variant of $x(w).\bar{w}z.P \xrightarrow{x(w)} \bar{w}z.P$. The transitions in the conclusion of each rule stand for all their variants. The Com_1 and $Close_1$ rules are obvious and are therefore omitted.

Just to show how enhanced labels are inductively built while deducing transitions, consider the process

$$(\nu\, x)(\bar{a}x.P_0 + \bar{b}y.P_1 \mid P_2) + P_3.$$

The derivation of the output of x on channel a starting from the axiom is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\bar{a}x.P_0 \xrightarrow{\bar{a}x} P_0}{\bar{a}x.P_0 + \bar{b}y.P_1 \xrightarrow{+_0\bar{a}x} P_0}\ Sum_0}{\bar{a}x.P_0 + \bar{b}y.P_1 \mid P_2 \xrightarrow{||_0+_0\bar{a}x} P_0 \mid P_2}\ Par_0}{(\nu\, x)(\bar{a}x.P_0 + \bar{b}y.P_1 \mid P_2) \xrightarrow{||_0+_0\bar{a}(x)} P_0 \mid P_2}\ Open}{(\nu\, x)(\bar{a}x.P_0 + \bar{b}y.P_1 \mid P_2) + P_3 \xrightarrow{+_0||_0+_0\bar{a}(x)} (P_0 \mid P_2) + P_3}\ Sum_0.$$

Hereafter, we write a transition as $P \xrightarrow{\theta} Q$ only if it is deductible according to the inference rules in Table 1; furthermore, we simply write it as θ, when it is unambiguous.

DEFINITION 2.3. A *proved transition system* is a quadruple $\langle \mathcal{P}, \Theta, \rightarrow, P_0 \rangle$, where $\mathcal{P}$ is the set of states (processes), Θ is the labelling alphabet, $\rightarrow$ is the transition relation defined in Table 1, and $P_0 \in \mathcal{P}$ is the initial state.

Standard semantics of the π-calculus is obtained by relabelling each transition shown in Table 1 by using function ℓ in Definition 2.2.

We now define proved computations.

DEFINITION 2.4. If $P_0 \xrightarrow{\theta} P_1$ is a transition, then P_0 is the *source* of the transition and P_1 is its *target*. A *proved computation* of P_0 is a sequence of transitions $P_0 \xrightarrow{\theta_0} P_1 \xrightarrow{\theta_1} \cdots$ such that the target of any transition is the source of the next one. We let ξ, ξ' range over proved computations. The notions of source and target are extended to computations.

3. STOCHASTIC SEMANTICS

We now show how to derive a probabilistic distribution F from a θ label. The intended meaning of F is the cost of execution (duration) of the action $\mu = \ell(\theta)$. The actual cost of μ depends on the basic operations that the run-time support of the target architecture performs for firing μ. For example, the resolution of a choice imposes various operations on the target architecture such as checking the ready list or implementing fairness policies. An action fired after a choice costs more than the same action occurring deterministically. The other operations of our calculus reflect analogous routines of the run-time support and delay the execution of an action as well. Therefore, we first assign a cost to the transition corresponding to μ on a dedicated architecture that only has to perform μ. We then model the performance degradation due to the run-time support by introducing a scaling factor for any routine implementing the transition. The new semantics takes into account the target architecture on which a system is run.

We derive the distributions of transitions by inspecting the syntactical contexts into which the actions which originate them are plugged. In fact, the context in which a μ action occurs represents the operations that the target machine performs for firing μ just because the structural operational semantics of a language specifies its abstract machine in a syntax-driven logical style. Accordingly, a linearization of a transition deduction (a proof term θ) represents the execution of the corresponding run-time support routines on the target machine.

For instance, look again at the sample deduction $\theta = +_0||_0 +_0 \overline{a}(x)$ reported at the end of the previous section. The enhanced label expresses that the abstract machine resolves two choices in favour of the left alternatives, thus adding extra costs to the output operation. Similarly, the selection of the left component of the parallel composition will have a cost depending on the allocation of processes and scheduling policies. The bound output means that the abstract machine has to handle the data structure representing the process environment in order to export the name x which is required to be fresh.

The tight connection of our cost model to the program syntax is a design choice. In fact, the way programs are written influences their performance. Consider two programs that only differ in some superfluous assignments or irrelevant loops. An example in our framework could be a P process that does not contain the name x and the $(\nu\, x)P$ process. A check on the name x is performed by the run-time support in the second process although x does not occur in P so that the two processes are behavioural equivalent, but $(\nu\, x)P$ is slower than P. To further support our design choice, we recall that context-free performance prediction can produce results that are inadequate an order of magnitude.

Following what is discussed above, we assign a cost to each inference rule of the operational semantics via a \$ function. In other words, the occurrence of a transition receives a duration time computed according to its deduction. The \$ function encodes the delay that the abstract machine adds to the execution of an action in order to handle the data structures of the run-time support of the language. Of course, the nearer the abstract machine to the target architecture, the more accurate the prediction of execution costs will be. Note that our approach allows us to estimate the performance of the *same program* on *different architectures* simply by changing the cost function. The classical comparison of *different programs* (but equivalent as far as functionalities are concerned) on the *same architecture* is possible as well by fixing a unique \$ for all of them.

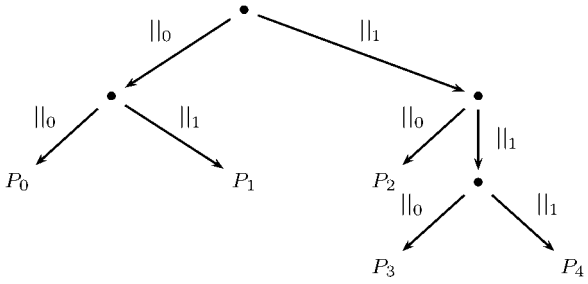


FIG. 1. Syntax tree of $(P_0 \mid P_1) \mid (P_2 \mid (P_3 \mid P_4))$.

There is no need to f x a \$ function here, and we let it be a parameter of the def nition of our model. For the sake of simplicity, we assume that $\$_a(\mu) = F_\mu \in \mathcal{F}$ (hereafter, we use $\mathcal{F}$ to denote a set of continuous probabilistic distribution functions) and that the slow-down factor is $\$_o(\vartheta) = r \in [1, +\infty)$. $\$_o$ can be def ned by inducing on the structure of ϑ. For instance, we can sum the slow-down factors of any $o \in \vartheta$ as

$$r = \$_o(\vartheta) = \sum_{o \in \vartheta} \$_o(o),$$

where

$$\$_o(||_i) = r_0, \quad \$_o(+_i) = r_1, \quad \$_o((\nu\, x)) = r_2, \quad \$_o(=_m) = r_3, \quad \$_o((\tilde{y})) = r_4,$$

where $r_i \in [1, +\infty)$, $i = 1,.., 4$. Of course, F_μ and r depend on architectural parameters as well. See [40, 41] for a complete def nition of $\$_o$ in the case of exponential distributions and [10] for the description of an implementation.

We now consider synchronizations. The two partners perform independently several low-level operations locally to their environment in order to set up the structures they need to communicate. These operations correspond to the rules applied to satisfy the premises of the *Com* and *Close* rules, which are recorded by pairing the proof terms corresponding to the local operations of the partners. Afterwards, in order to pass through the context common to the two partners, a few operations are necessary. The cost of these additional operators is derived using $\$_o$. For instance, consider the process

$$R + (((\nu\, x)(\overline{a}x + \overline{b}y)) \mid ((a(x) \mid b(x)) + Q)).$$

The communication along link a has the label $+_1\langle ||_0 +_0 \overline{a}(x), ||_1 +_0 ||_0 a(x)\rangle$. The two components of the pair are the local operations of the partners, while the leftmost $+_1$ is a common operation performed after the $Close_0$ rule has been applied.

We then use the function $f_{\langle\rangle} : \mathcal{L}^* \times \mathcal{L}^* \to [1, +\infty)$ to take the distance of the two partners into account. For encoding locations, we use $\chi_0, \chi_1 \in \{||_0, ||_1\}^* = \mathcal{L}^*$. To understand why, consider the binary abstract syntax tree of a process, where the parallel composition | is the only syntactic operator. Then, a sequence χ is the access path from the root (the whole process) to a leaf (a subprocess). Consider for instance $(P_0 \mid P_1) \mid (P_2 \mid (P_3 \mid P_4))$ and its syntax tree shown in Fig. 1. The access path $||_1||_1||_0$ from the root uniquely identif es the process P_3, while $||_0||_1$ is the path for P_1. Therefore, the allocation of processes can be described by $All : \mathcal{L}^* \to Loc$, where Loc is the set of physical locations. We do not explicitly introduce the function All here for the sake of presentation. Actually, All is only a degree of indirection from strings in $\mathcal{L}^*$ (that hereafter by abuse of wording we call locations). The two arguments of $f_{\langle\rangle}$, together with allocation tables, can be used to determine where the two communicating processes actually reside. Note that binary trees are only used to generate names of locations and they are completely independent of the topology of the interconnection network.

To apply function $f_{\langle\rangle}$, we need an auxiliary function $\underline{\cdot} : (\mathcal{L} \cup \mathcal{O})^* \to \mathcal{L}^*$ that extracts the parallel tags from proof terms, inductively def ned as (ϵ is the empty string)

$$\underline{\epsilon} = \epsilon, \quad \underline{o\mathfrak{A}} = \begin{cases} ||_i \underline{\mathfrak{A}} & \text{if } o = ||_i \\ \underline{\mathfrak{A}} & \text{otherwise.} \end{cases}$$

The tags that we discard from enhanced labels only specify characteristics of sequential components of the system. Thus they are not relevant for allocation tables.

Hereafter, we use f for the density function corresponding to the distribution function F of a continuous random variable. Remember that the relation between density and distribution functions is $\int_{\infty}^{x} f(t)\,dt = F(\overline{x})$, or equivalently $F'(t) = f(t)$, where F' is the derivative of F.

Function \$ is def ned by using the auxiliary functions $\$_a$ as the basis along with $\$_o$ and $f_{\langle\rangle}$. To slow down the f ring time of the transitions, we use a homotety[2] on the arguments of the distribution functions. In other words, given a function $F(x)$, we def ne $G(x) = F(x/t)$ with $t \in \mathbb{R}^+$. When F is a distribution function, $\lim_{x\to+\infty} F(x) = 1$ and the speed at which F approaches its asymptotic value 1 makes the time interval described by a random variable T with distribution F vary. The faster F approaches 1, the smaller the time interval, and hence the faster the transition corresponding to T. The homotety x/t makes G faster than F when $t < 1$ and G slower than F when $t > 1$. This is why we let the codomain of $\$_o$ and $f_{\langle\rangle}$ be $[1, +\infty)$.

Note that one may resort to density functions $f(x)$ to slow down transition speed. The idea is to increase the expectation of the time interval which is $EX_f = \int x f(x)\,dx$. We still apply the homotety x/t, yielding a function $f_t(x) = f(x/t)$. The expectation becomes

$$EX_{f_t} = \int x f_t(x)\,dx = t\int (x/t) f(x/t)\,dx/t = tEX_f.$$

This result agrees with the homotety applied to distribution functions because $t > 1$ increases the expectation and thus slows down the corresponding transition.

We can now def ne our cost function \$.

Definition 3.1. Given a set $\mathcal{F}$ of continuous probabilistic distributions, the function $\$: \Theta \to \mathcal{F}$ is def ned as follows:

$$\$(\vartheta\,\mu) = F\left(\frac{x}{\$_o(\vartheta)}\right) \qquad \text{if } \$_a(\mu) = F(x),$$

$$\$(\vartheta\,\langle \vartheta_0\alpha_0, \vartheta_1\alpha_1\rangle) = F\left(\frac{x}{\$_o(\vartheta) \bullet f_{\langle\rangle}(\mathfrak{A}_0, \mathfrak{A}_1)}\right) \qquad \text{if } \min\{\$(\vartheta_0\alpha_0), \$(\vartheta_1\alpha_1)\} = F(x),$$

where $\bullet : [1, +\infty) \times [1, +\infty) \to [1, +\infty)$ is monotonic and such that $x \bullet y \geq z$ with $z \in \{x, y\}$.

For instance, the sum and multiplication on $\mathbb{R}$ satisfy the requirements for $\bullet$.

Note that determining the distribution of synchronizations is a key point in distinguishing different proposals of stochastic process algebras. Here we demand its computation to function \$. This means that the distributions of synchronization vary according to the context into which they are plugged and to the architectures on which the partners run. We thus reduce the selection of distributions to the selection of suitable architectures and placement of processes. This way the designer may abstract from stochastic details and concentrate on the characteristics of the hardware. Examples of application of this mechanism can be found in [10, 40].

Since we are dealing with performance evaluation, we need to eliminate the nondeterminism introduced by the choice operator from stochastic transition systems. Hence, we introduce a *race condition* that selects the transition to be f red among the ones enabled in a state. All the enabled transitions attempt to proceed, but only the fastest one succeeds. This mechanism makes the nondeterministic choice a probabilistic one. Note that the continuous nature of probabilistic distributions ensures that the probability of two transitions ending simultaneously is 0. Moreover, as the duration of transitions is expressed by random variables, different transitions are selected on different attempts.

Hereafter the apparent rate of an action a in P (written $r_a(P)$) will be the rate captured by an external observer of the system, which can only register actions and their occurrence frequency (for a formal

[2] Given a vector space V on $\mathbb{R}$ and $t \in \mathbb{R}$, $t > 0$, the function $tI : V \to V$ with $x \mapsto tx$ is a homotety. If V is equipped with a scalar product (e.g., $\mathbb{R}^n$), a homotety with $t < 1$ ($t > 1$) makes the distance between any two points in the space smaller (greater). By abuse of wording, sometimes the term homotety is used to refer to the constant t, as well. In our setting, we use the constant t to tune the arguments of distribution functions and we still call the transformation a homotety.

def nition see the theorem below) [30]. Also, T_i denotes the random variable describing the time interval associated with a $P \xrightarrow{\theta_i} P_i$ transition. The following theorem shows how to compute useful probabilities and transition distributions.

THEOREM 3.1. *The probability of* $P \xrightarrow{\theta_i} P_i$ *is*

$$p_i = \int_0^\infty f_i(t) \cdot \prod_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} (1 - \$(\theta_j)(t))\, dt;$$

the distribution of the random variable T_i *which describes the time interval associated with* $P \xrightarrow{\theta_i} P_i$ *is*

$$\tilde{F}_i(t) = \frac{\int_0^t f_i(x) \cdot \prod_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} (1 - \$(\theta_j)(x))\, dx}{p_i};$$

the apparent rate of an a action in P *is*

$$r_a(P) = \sum_{\substack{P \xrightarrow{\theta_i} P_i \\ \ell(\theta_i) = a}} p_i;$$

the probability of $P \xrightarrow{\theta_i} P_i$, $\ell(\theta_i) = a$, *given that an a action occurred, is*

$$\frac{p_i}{r_a(P)}.$$

Proof. By race condition, the probability that $P \xrightarrow{\theta_i} P_i$ occurs is $\mathcal{P}r(\bigwedge_{i \neq j} T_i < T_j)$. To compute this probability, consider the conditional probability distribution

$$F_{T_i|T_j} = \mathcal{P}r\left(\bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} T_i < T_j \mid T_i = t \right) = \mathcal{P}r\left(\bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} T_j > t \mid T_i = t \right)$$

by independence of the variables T_j

$$\mathcal{P}r\left(\bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} T_j > t \right) = \prod_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} \left(1 - \$(\theta_j)(t)\right).$$

Then, the probability of $P \xrightarrow{\theta_i} P_i$ is given by the continuous version of the theorem of total probability

$$\int_0^\infty f_i(t) \cdot \prod_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} (1 - \$(\theta_j)(t))\, dt = p_i.$$

Now consider the second statement. By def nition of the distribution function,

$$\tilde{F}_i(t) = \mathcal{P}r\left(T_i \leq t \mid \bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} T_i < T_j \right)$$

by the continuous analog of Bayes' rule

$$\frac{\int_0^t f_i(x) \cdot \mathcal{P}r\left(\bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} T_i < T_j \mid T_i \leq t \right) dx}{\int_0^\infty f_i(x) \cdot \mathcal{P}r\left(\bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} T_i < T_j \mid T_i \leq t \right) dx}.$$

We can write $\mathcal{P}r(\bigwedge_{i \neq j} T_i < T_j \mid T_i \leq t)$ as $\mathcal{P}r(\bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} T_j > x)$, and by independence of the variables T_j, we obtain

$$\mathcal{P}r\left(\bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} T_j > x \right) = \prod_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} \mathcal{P}r(T_j > x) = \prod_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} (1 - \$(\theta_j)(x)),$$

from which the thesis follows.

The apparent rate in the statement can be written as

$$r_a(P) = \bigcup_{\substack{P \xrightarrow{\theta_i} P_i \\ \ell(\theta_i) = a}} \mathcal{P}r\left(\bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} T_i < T_j \right)$$

because $\forall i \neq l \,.\, (\bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} T_i < T_j) \cap (\bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ l \neq j}} T_l < T_j) = \emptyset$

$$\sum_{\substack{P \xrightarrow{\theta_i} P_i \\ \ell(\theta_i) = a}} \mathcal{P}r\left(\bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} T_i < T_j \right) = \sum_{\substack{P \xrightarrow{\theta_i} P_i \\ \ell(\theta_i) = a}} p_i.$$

The conditional probability of the statement can be written as

$$\mathcal{P}r\left(\bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} T_i < T_j \mid \bigvee_{\substack{P \xrightarrow{\theta_h} P_i \\ \ell(\theta_h) = a}} \left(\bigwedge_{\substack{P \xrightarrow{\theta_j} P_j \\ h \neq j}} T_h < T_j \right) \right)$$

by def nition of conditional probability

$$\frac{\mathcal{P}r\left(\bigwedge_{\substack{P\xrightarrow{\theta_j}P_j\\ i\neq j}} T_i < T_j \wedge \bigvee_{\substack{P\xrightarrow{\theta_h}P_i\\ \ell(\theta_h)=a}} \left(\bigwedge_{\substack{P\xrightarrow{\theta_j}P_j\\ h\neq j}} T_h < T_j\right)\right)}{\mathcal{P}r\left(\bigvee_{\substack{P\xrightarrow{\theta_h}P_i\\ \ell(\theta_h)=a}} \left(\bigwedge_{\substack{P\xrightarrow{\theta_j}P_j\\ h\neq j}} T_h < T_j\right)\right)}$$

because T_i is associated with an a-transition and because the denominator is $r_a(P)$

$$\frac{\mathcal{P}r\left(\bigwedge_{\substack{P\xrightarrow{\theta_j}P_j\\ i\neq j}} T_i < T_j\right)}{r_a(P)} = \frac{p_i}{r_a(P)}. \qquad \blacksquare$$

To get performance measures from our transition systems, we must update the distributions of the random variables that express the time interval associated with transitions in correspondence with branching points. We also compute the occurrence probability for any transition and record it in the transition labels. The f rst two statements of Theorem 3.1 allow us to compute the correct occurrence probabilities and distributions. The new transition system is called stochastic.

DEFINITION 3.2. The quadruple $\langle \mathcal{P}, \Theta \times \mathcal{F} \times [0..1], \rightarrow, P_0\rangle$ is the *stochastic transition system* associated with process P_0, where the real in $[0..1]$ denotes transition occurrence probability. The relation $\rightarrow$ is def ned as

$$\frac{P \xrightarrow{\theta_i} P_i}{P \xrightarrow{\theta_i, \tilde{F}_i, p_i} P_i},$$

where

$$\tilde{F}_i = \frac{\int_0^t f_i(x) \cdot \prod_{\substack{P\xrightarrow{\theta_j}P_j\\ i\neq j}} (1 - \$(\theta_j)(x))\, dx}{\int_0^\infty f_i(x) \cdot \prod_{\substack{P\xrightarrow{\theta_j}P_j\\ i\neq j}} (1 - \$(\theta_j)(x))\, dx},$$

and

$$p_i = \int_0^\infty f_i(t) \cdot \prod_{\substack{P\xrightarrow{\theta_j}P_j\\ i\neq j}} (1 - \$(\theta_j)(t))\, dt.$$

The labels distinguish stochastic transitions from proved ones.

4. ENABLING MEMORY DISCIPLINE

We now introduce a relation of enabling between the transitions of a computation simply by looking at their labels [20]. This relation between transitions will be used in the next section to deal with general distributions when the random variables associated with the transitions of a computation are not independent. Hereafter we write ϑ in place of $\mathfrak{d}$ for the sake of readability. Also, we def ne the enabling relation on proved computations. Its extension to stochastic computations only amounts to applying the def nitions to the f rst component of the stochastic transition labels. Consequently, enabling could be

computed simultaneously with stochastic information. We prefer to present it as a further step simply to improve readability.

4.1. Enabling and Concurrency

We first define structural dependencies. A transition labelled $\vartheta\mu$ depends on a previous transition labelled $\vartheta'\mu'$ if ϑ' is a prefix of ϑ (the tuning needed to cover communications is explained below). The underlying idea is that the two transitions have been derived using the same initial set of rules and are thus nested in a prefix chain (or they are connected by communications in a similar way).

DEFINITION 4.1. If $P_0 \xrightarrow{\theta_0} P_1 \xrightarrow{\theta_1} \cdots \xrightarrow{\theta_n} P_{n+1}$ is a proved computation, and hereafter $i, j \in \{0, 1\}$, then θ_n has a *direct structural dependency* on θ_h, $h < n$, $(\theta_h \preceq^1_{str} \theta_n)$ iff

- $\theta_n = \vartheta\mu, \theta_h = \vartheta'\mu'$ and ϑ' is a prefix of ϑ; or
- $\theta_n = \vartheta\mu, \theta_h = \vartheta'\langle\vartheta'_0\mu'_0, \vartheta'_1\mu'_1\rangle$ and $\exists i\,.\,\vartheta'\vartheta'_i$ is a prefix of ϑ; or
- $\theta_n = \vartheta\langle\vartheta_0\mu_0, \vartheta_1\mu_1\rangle, \theta_h = \vartheta'\mu', \exists i\,.\,\vartheta'$ is a prefix of $\vartheta\vartheta_i$; or
- $\theta_n = \vartheta\langle\vartheta_0\mu_0, \vartheta_1\mu_1\rangle, \theta_h = \vartheta'\langle\vartheta'_0\mu'_0, \vartheta'_1\mu'_1\rangle, \exists i, j\,.\,\vartheta'\vartheta'_j$ is a prefix of $\vartheta\vartheta_i$.

The *structural dependencies* of θ_n are obtained by reflexive and transitive closures of $\preceq^1_{str}$, i.e., $\preceq_{str} = (\preceq^1_{str})^*$.

The last two items in Definition 4.1. say that a θ transition enables a communication if it enables one of its components. Also, we need the transitive closure of $\preceq^1_{str}$ to implement the cross inheritance of the causes of the communication partners for the residual processes.

Consider the process

$$P_0 = (\nu b)(a.\overline{b} \mid d.b.(\nu z)(\overline{x}z \mid \overline{z}z))$$

and its proved computation (here only labelled with tags $||_i$; $(\nu\ b)$, which prefixes all labels, is omitted)

$$P_0 \xrightarrow{||_0 a} P_1 \xrightarrow{||_1 d} P_2 \xrightarrow{\langle||_0\overline{b}, ||_1 b\rangle} P_3 \xrightarrow{||_1||_0\overline{x}z} P_4 \xrightarrow{||_1||_1\overline{z}z} (\nu b)(\mathbf{0}|(\mathbf{0}|\mathbf{0})). \tag{1}$$

Since $||_1$, the proof term of b, is a prefix of the proof term $||_1||_0$ of the bound output, $\overline{x}z$ depends on the communication; thus it also inherits the causes of the sender (the first transition). The bound output $\overline{x}(z)$ also depends on the second transition whose proof term is a prefix of its own.

The second step defines name dependencies that are only generated by bound outputs. This is because a name dependency between an input, which binds a name y, and its following usage always induces a structural dependency as well (see [20] for more details).

DEFINITION 4.2. If $P_0 \xrightarrow{\theta_0} P_1 \xrightarrow{\theta_1} \cdots \xrightarrow{\theta_n} P_{n+1}$ is a proved computation, then the *name enabler* of θ_n, if any, is (the unique) θ_h $(\theta_h \prec_{nam} \theta_n)$ such that $\ell(\theta_h) = \overline{x}a$, $\forall j\,.\,h < j < n, a \notin bn(\ell(\theta_j))$, and $\ell(\theta_n) \in \{\overline{a}z, \overline{a}z, a(z), \overline{y}a\}$.

Look again at computation (1). The output on link z depends on the bound output, as z has been extruded by $\overline{x}z$.

All the dependencies of a transition are the union of its structural dependencies, its name dependency θ, and the set containing the name and the structural dependencies of θ. Thus the *enabling* relation is

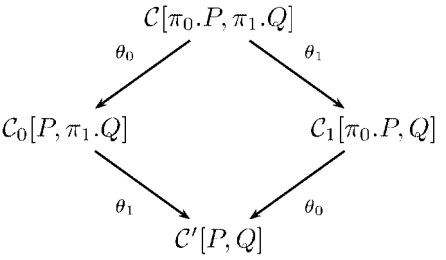
$$\preceq = (\preceq_{str} \cup \prec_{nam})^*.$$

We also need the definition of a concurrency relation $(\frown)$ between transitions that we recall from [20]. Roughly, two transitions are concurrent if they result from firing two prefixes lying on opposite sides of a $|$ and there is no way of sequentializing them.

DEFINITION 4.3. If $P_0 \xrightarrow{\theta_0} P_1 \xrightarrow{\theta_1} \cdots \xrightarrow{\theta_n} P_{n+1}$ is a proved computation, then the *concurrency* relation between transitions is $\frown$ such that $\theta_h \frown \theta_n \Leftrightarrow \theta_h \not\preceq \theta_n$.

The above definition allows us to form the following theorem, which is proven in [20].

THEOREM 4.1. *If* $\mathcal{C}[\bullet, \bullet]$, $\mathcal{C}'[\bullet, \bullet]$, $\mathcal{C}_0[\bullet, \bullet]$, *and* $\mathcal{C}_1[\bullet, \bullet]$ *are* (*nonempty*) *contexts with* (*exactly*) *two holes*, *then the proved transition system contains the diamond*



with actions θ_0 (θ_1) *originated by the* same *prefix* π_0 (π_1) *if and only if* $\theta_0 \smile \theta_1$.

4.2. Distribution of Transitions

We now use enabling to keep track of the history of a system to determine the firing distribution of transitions. Here we use the *enabling memory discipline with race* defined for stochastic Petri nets [1] and used in TIPP stochastic process algebra [24] (although implemented with a different mechanism; see below). The firing distribution of a transition accounts for the work performed by the corresponding action once it becomes enabled.

Consider, for instance, the process $a|b$ and its computation

$$a|b \xrightarrow{||_0 a} \mathbf{0}|b \xrightarrow{||_1 b} \mathbf{0}|\mathbf{0}.$$

Since the transition $a|b \xrightarrow{||_1 b} a|\mathbf{0}$ is possible as well because $||_0 a \not\preceq ||_1 b$, and the selection of the one labelled a does not prevent b from occurring, the random variable describing the time interval T_0 associated with the first transition is not independent on T_1 associated with the second one. In fact, the time interval of the b-transition in the computation above overlaps with and continues after T_0. Thus, according to the enabling memory discipline, the variable T_1 must account for the time elapsed during the firing of a, plus the extra time up until the completion of b.

In [1], the enabling memory discipline with race is implemented through so-called age variables associated with transitions. Instead, [24] modifies the syntax of the language (and hence its semantics) in order to keep track of the number of time intervals during which a transition was enabled, but a concurrent one was selected for firing. We show here that we do not need to modify the syntax of the π-calculus or the states of its transition system to handle general distributions.

Let $P_0 \xrightarrow{\theta_0, F_0, p_0} P_1 \xrightarrow{\theta_1, F_1, p_1} \cdots \xrightarrow{\theta_n, F_n, p_n} P_{n+1}$ be a stochastic computation, and assume that one wants to determine the firing distribution of θ_n. Since mutual exclusive (or conflicting) transitions are represented by branching points in the structure of transition systems, any pair of transitions in a given computation is either in an enabling or in a concurrency relation (see Definition 4.3 and also [20]). Therefore, we only need to find the maximum i ($0 \leq i < n$), if any, such that $\theta_i \preceq \theta_n$. Then, all the transitions following θ_i and preceding θ_n are concurrent with θ_n. (Note that there is at least one such transition if $i < n - 1$.) All the time periods in which these transitions occurred must influence the firing distribution of θ_n. We first need an auxiliary definition of immediate dependency. A transition θ_i immediately enables θ_n iff there is no transition between θ_i and θ_n that enables θ_n. We denote the new relation $\preceq_c$ because it is the covering relation of $\preceq$.

DEFINITION 4.4. If $P_0 \xrightarrow{\theta_0, F_0, p_0} P_1 \xrightarrow{\theta_1, F_1, p_1} \cdots \xrightarrow{\theta_n, F_n, p_n} P_{n+1}$ is a stochastic computation, then θ_i *immediately enables* θ_n ($\theta_i \preceq_c \theta_n$) iff $\theta_i \preceq \theta_n$, and $\forall j.\ i < j < n, \theta_j \not\preceq \theta_n$.

Note that the new relation $\preceq_c$ only differs from $\preceq^1_{str}$ because the latter relation can select transitions which are not the *immediate* enablers of the current one. For instance, consider computation (1) again. It is $||_1 d \preceq^1_{str} ||_1 ||_0 \overline{x}(z)$, but $||_1 d \not\preceq_c ||_1 ||_0 \overline{x}(z)$.

We can now define the distribution of the random variable T_n associated with θ_n.

DEFINITION 4.5. If $P_0 \xrightarrow{\theta_0, F_0, p_0} P_1 \xrightarrow{\theta_1, F_1, p_1} \cdots \xrightarrow{\theta_n, F_n, p_n} P_{n+1}$ is a stochastic computation, then the distribution of T_n is

$$\mathcal{P}r\left(T_n \leq t + \sum_{h=i+1}^{n-1} T_h \mid T_n > \sum_{h=i+1}^{n-1} T_h\right) \quad \text{with } \theta_i \preceq_c \theta_n$$

assuming as usual that $\sum_{\emptyset} T_h = 0$.

As described in [20], many semantic models (causal, local, interleaving) suitable for behavioral analysis can be retrieved from the proved transition system of π-calculus through simple relabelling functions of transition labels. Here we apply the same idea to yield a transition system suitable for performance analysis by relabelling stochastic computations.

We apply the relabelling function to transition θ_n only after relabelling transitions θ_i, and $0 \leq i < n$. This is necessary to correctly compute the distribution of $\sum_{h=i+1}^{n-1} T_h$. We can thus inductively relabel any transition of a stochastic computation by replacing the distribution of the transition with the one computed according to Def nition 4.5. Once again, note that the distributions of transitions can be directly computed while building the stochastic transition system simply by composing what follows with the calculations described in Def nition 3.2. We think that this separation of concerns can help to understand or make things clearer.

DEFINITION 4.6. If $\xi_0 = P_0 \xrightarrow{\theta_0, F_0, p_0} P_1 \xrightarrow{\theta_1, F_1, p_1} \cdots \xrightarrow{\theta_n, F_n, p_n} P_{n+1}$ is a stochastic computation, the corresponding *performance computation* is

$$\xi_{n+1} = P_0 \xrightarrow{\theta_0, F_0'} P_1 \xrightarrow{\theta_1, F_1'} \cdots \xrightarrow{\theta_n, F_n'} P_{n+1}$$

and is obtained by applying the function $S_n \circ S_{n-1} \circ \cdots \circ S_0$ to ξ_0 with

$$S_i(\xi_i) = \xi_{i-1}\left\{\left(P_i \xrightarrow{\theta_i, F_i'} P_{i+1}\right) \Big/ \left(P_i \xrightarrow{\theta_i, F_i, p_i} P_{i+1}\right)\right\},$$

where $F_i' = \mathcal{P}r(T_i \leq t + \sum_{h=j+1}^{i-1} T_h \mid T_i > \sum_{h=j+1}^{i-1} T_h)$, $j < h < i, \theta_j \preceq_c \theta_i$.

4.3. Stochastic Processes

In this section we show how to raise the relabelling of stochastic transitions from computations to stochastic transition systems.

The second f ring of the same transition in a computation can be interpreted as a resampling of an experiment. Of course, two distinct experiments must be done independent of one another to be meaningful. In fact, under the f ring policy of race with enabling memory, a new delay has to be sampled if a disabled transition becomes re-enabled (see also [1]). Consequently, the history of the system that leads to a $P \xrightarrow{\theta, F, p} P'$ transition must not be taken into consideration to compute the distributions of its following occurrence.

We f rst need an auxiliary def nition to identify the computations whose transitions occur once at the most, and their source state is the initial state of the transition system.

DEFINITION 4.7. If $ts = \langle \mathcal{P}, \Theta \times \mathcal{F} \times [0..1], \rightarrow, P_0 \rangle$ is a stochastic transition system and P_n one of its states, then the set of the *acyclic computations* from P_0 to P_n is

$$AC(ts, P_n) = \{\xi_i \mid \text{source}(\xi_i) = P_0, \text{target}(\xi_i) = P_n, P_n \xrightarrow{\theta_n, F_n, p_n} P_{n+1} \notin \xi_i,$$
$$\forall P_j \xrightarrow{\theta_j, F_j, p_j} P_{j+1}, P_k \xrightarrow{\theta_k, F_k, p_k} P_{k+1} \in \xi_i\ .$$
$$j \neq k \Rightarrow P_j \xrightarrow{\theta_j, F_j, p_j} P_{j+1} \neq P_k \xrightarrow{\theta_k, F_k, p_k} P_{k+1}\}.$$

The following theorem establishes that a finite state transition system (that may have loops) gives origin to a finite set of finite acyclic computations.

THEOREM 4.2. *If* $ts = \langle \mathcal{P}, \Theta \times \mathcal{F} \times [0..1], \rightarrow, P_0 \rangle$ *is a finite state stochastic transition system, then*

$$\forall P \in ts\,.\, AC(ts, P) \text{ is finite and } \forall \xi \in AC(ts, P)\,.\, \xi \text{ is finite.}$$

Proof. Since the proved transition system is finite branching (see the rules in Table 1), the stochastic transition system is finite branching as well. In fact, we define it by relabelling proved transitions. Hence, a finite state transition system has a finite number of transitions. So they are the combination of its transitions as well as the length of its acyclic computations. ■

Hereafter, we use $\mathcal{P}r(\xi)$ to denote the occurrence probability of the last transition of ξ, i.e.,

$$\mathcal{P}r\Big(\xi = P_0 \xrightarrow{\theta_0, F_0, p_0} P_1 \xrightarrow{\theta_1, F_1, p_1} \cdots \xrightarrow{\theta_n, F_n, p_n} P_{n+1}\Big) = \prod_{i=0}^{n} p_i.$$

To compute the distribution of a $P_n \xrightarrow{\theta_n, F_n, p_n} P_{n+1}$ transition under the firing policy of race with enabling memory, we rely on the set $AC(ts, P_n)$. We apply Definition 4.6 to $\xi_i \xrightarrow{\theta_n, F_n, p_n} P_{n+1}$ for any $\xi_i \in AC(ts, P_n)$, which yields $F_n^i\ \alpha$ as a distribution for θ_n. We eventually obtain the distribution of θ_n under the enabling memory discipline by mediating the F_n^i's through the occurrence probabilities of θ_n as a last transition of the computations in $AC(ts, P_n)$.

DEFINITION 4.8. If $ts = \langle \mathcal{P}, \Theta \times \mathcal{F} \times [0..1], \rightarrow, P \rangle$ is a stochastic transition system, the corresponding *performance transition system* $\langle \mathcal{P}, \mathcal{A} \cup \{\tau\} \times \mathcal{F}, \rightarrow, P \rangle$ is obtained by relabelling any transition $P_n \xrightarrow{\theta_n, F_n, p_n} P_{n+1}$ as $P_n \xrightarrow{\theta_n, F_n'} P_{n+1}$ where

$$F_n' = p_n \times \sum_{\xi_i \in AC(ts, P_n)} \mathcal{P}r(\xi_i) \times F_n^i$$

with F_n^i as the distribution associated with $P_n \xrightarrow{\theta_n, F_n, p_n} P_{n+1}$, by applying Definition 4.6 to $\xi_i \xrightarrow{\theta_n, F_n, p_n} P_{n+1}$.

Note that the above definition is effective for finite state transition systems according to Theorem 4.2. Furthermore, performance transition systems can be inductively derived from processes without relying on stochastic transition systems (see the discussion immediately before Definition 4.6; the idea is to build the transition system from a breadth-first strategy and treat the generation of all the transitions exiting a state as atomic).

The enabling memory discipline as defined in this section, though it coincides with some proposals in the literature [1, 24], has certain limitations. The next section introduces a refined discipline.

5. REFINING THE ENABLING MEMORY DISCIPLINE

As announced in the Introduction, here we use a different notion of timer associated with transitions. We are not only interested in the CPU consumption of transitions, but also in their response time. Therefore, we let our timers count the time during which transitions are enabled and not scheduled on a CPU as well.

Consider the proved computation again

$$a|b \xrightarrow{||_0 a} \mathbf{0}|b \xrightarrow{||_1 b} \mathbf{0}|\mathbf{0}.$$

The random variables T_0 and T_1 associated with $||_0 a$ and $||_1 b$ are considered dependent in Section 4.2 because the time period of $||_1 b$ overlaps with that of $||_0 a$. The motivation was that $||_0 a$ and $||_1 b$ are concurrent transitions; thus their execution ordering is irrelevant from a behavioral point of view. This

is certainly true when we run the system on a single processor. Hence, concurrency is simulated by interleaving concurrent transitions and the dependency of the variables shows up.

Now consider a network of at least two machines and run $a|b$ on it. A suitable allocation of subprocesses permits a truly concurrent execution of $||_0 a$ and $||_1 b$ that can be represented by the transition $a|b \xrightarrow{||_0 a, ||_1 b} \mathbf{0}|\mathbf{0}$. In this case, the dependency of the random variables T_0 and T_1 is no longer valid. In fact, the two transitions are executed by different and *independent* machines. Thus it is evident that the number of available resources inf uences the stochastic structure of the system at hand. We can generalize the dual machine network to any number n of nodes without modifying our discussion. We only need to resort to transitions that carry up to n enhanced labels.

Note that communications impose synchronization points between independent threads of computation. Consider for instance the system $(\nu c)(a.c.d \mid b.\bar{c}.e)$ where the communication over c imposes a synchronization between the concurrent executions of a with b and of d with e. This synchronization works independent of the number of processors and therefore it is imposed by the structure of the system.

Given the operational semantics of the π-calculus, we can discuss how to generate transitions with multiple labels. We only need to label those transitions carrying a single θ with a singleton. Given a set of labels I, we write $\theta \in I$ for the more precise, yet verbose, $\langle \theta, F, p \rangle \in I$. To improve readability, we write $\iota(\theta)$ for the distribution F associated with θ in the labels of performance computations.

DEFINITION 5.1. Assuming that standard labels are actually singletons, the following rule composes transitions labelled by sets of actions to yield higher-dimension transitions

$$\frac{P \xrightarrow{I_0} P',\ P' \xrightarrow{I_1} Q,\ I_0 \smile I_1}{P \xrightarrow{I_0 \cup I_1} Q}, \quad |I_0 \cup I_1| \leq n,$$

where n is the number of processors available, $|I|$ the cardinality of the set I, and I_0 and I_1 sets of labels with

$$I_0 \smile I_1 \Leftrightarrow \forall \theta_0 \in I_0, \forall \theta_1 \in I_1,\ \theta_0 \smile \theta_1.$$

Of course, the above def nition can be applied to stochastic and performance transitions as well. The same strategy used to generate performance transition systems from processes applies here to generate higher-dimension transition systems as well as their stochastic and performance versions.

As an example, consider the computation (1) again which, by allowing higher dimension transitions, becomes

$$P_0 \xrightarrow{\{||_0 a, ||_1 d\}} P_2 \xrightarrow{\{\langle ||_0 \bar{b}, ||_1 b \rangle\}} P_3 \xrightarrow{\{||_1 ||_0 \bar{x} z\}} P' \xrightarrow{\{||_1 ||_1 \bar{z} z\}} P_6$$

if at least two processors are available.

We now modify Def nition 4.6 to compute the distributions of the higher dimension transitions. Consider the higher dimension stochastic computation ξ

$$P_0 \xrightarrow{I_0} P_1 \xrightarrow{I_1} \cdots \xrightarrow{I_n} P_{n+1}$$

and assume that one wants to determine the f ring distribution of $\theta' \in I_n$. According to the enabling memory discipline, we must look for a $\theta \in I_i$ with $\theta \preceq_c \theta'$. Then, the activities belonging to any I_l, where $i < l < n$, are all concurrent with θ'. However, we do not need to consider the time elapsed for all the activities in I_l to compute the distribution of θ'. For any set I_l, we only need to consider the slowest activity. Since all θ's in I_l are executed independently, the time period of the slowest one is given as

$$\Pi_{\theta \in I_l}(1 - \iota(\theta)(x)).$$

Note that this is a simplif cation. Indeed, the execution of θ' can partially overlap with the one with the slowest activity in I_l, if it does not enable θ'. For instance, consider $a.b|c.d|e$ and its higher dimension computation

$$a.b|c.d|e \xrightarrow{\{a,c,e\}} b|d|\mathbf{0} \xrightarrow{\{b,d\}} \mathbf{0}|\mathbf{0}|\mathbf{0}.$$

The execution of d in the second transition can overlap with that of a, if a is much slower than c. What we get with our assumption is actually an upper bound to the response time of the system. Finally, note that this assumption can be used to work with synchronous systems as well.

DEFINITION 5.2. If $P_0 \xrightarrow{I_0} P_1 \xrightarrow{I_1} \cdots \xrightarrow{I_n} P_{n+1}$ is a higher dimension stochastic computation, then the distribution of the random variable T' associated with $\theta' \in I_n$ is

$$\mathcal{P}r\left(T' \leq t + \sum_{h=i+1}^{n-1} \hat{T}_h \mid \hat{T}_h \text{ has distribution } \Pi_{\theta \in I_l}(1 - \iota(\theta)(x)) \text{ and } T' > \sum_{h=i+1}^{n-1} \hat{T}_h\right),$$

where $\theta \in I_i$, $\theta \preceq_c \theta'$.

We can now adapt Def nition 4.6 to higher dimension transitions. We only need to change the def nition of $S_i(\xi_i)$.

DEFINITION 5.3. If $\xi_0 = P_0 \xrightarrow{I_0} P_1 \xrightarrow{I_1} \cdots \xrightarrow{I_n} P_{n+1}$ is a higher dimension stochastic computation, the corresponding higher dimension performance computation $\xi_{n+1} = P_0 \xrightarrow{I'_0} P_1 \xrightarrow{I'_1} \cdots \xrightarrow{I'_n} P_{n+1}$ is obtained by applying the function $S_n \circ S_{n-1} \circ \cdots \circ S_0$ to ξ_0 where

$$S_i(\xi_i) = \xi_{i-1}\left\{\left(P_i \xrightarrow{I'_i} P_{i+1}\right) \Big/ \left(P_i \xrightarrow{I_i} P_{i+1}\right)\right\},$$

where $I'_i = \{\theta_j, F'_j \mid \theta_j \in I_i\}$ and F'_j is def ned according to Def nition 5.2.

5.1. Unbound Processors

Here we study the cases where an unbound number of processors is available. We call this hypothesis *maximal parallelism assumption*. The weaker assumption of having a number of processors greater than or equal to the maximum number of simultaneously enabled transitions works as well. We can further weaken the assumption by assuming as many processors as the cardinality of the greater subset of enabled transitions such that no two transitions in the set are conf icting. This set identif es the maximal number of transitions that can be executed simultaneously and independently.

We now report a property of higher dimension transition systems (we omit the distributions of transitions in the labels). We write $\rightarrow^*$ for the ref exive and transitive closures of a transition relation $\rightarrow$.

PROPOSITION 5.1. *Given a* $P_0 \xrightarrow{\{\theta_1,\ldots,\theta_n\}} P_{n+1}$ *transition in a higher dimension transition system, there are* $n!$ *computations* $P_0 \rightarrow^* P_{n+1}$ *whose transitions are labelled by the* $n!$ *permutations of the* $\{\theta_1\}, \ldots, \{\theta_n\}$ *singletons.*

Proof. From $P_0 \xrightarrow{\{\theta_1,\ldots,\theta_n\}} P_{n+1}$ we obtain $P_0 \xrightarrow{\{\theta_1\}} P_1 \xrightarrow{\{\theta_2\}} \cdots \xrightarrow{\{\theta_n\}} P_{n+1}$ by repeated decompositions of the premises of the rule in Def nition 5.1. Then, we apply Theorem 4.1 to any pair of consecutive concurrent transitions for as long as possible. Since any pair of enhanced labels in $\{\theta_1, \ldots, \theta_n\}$ describes concurrent transitions, we can build the $n!$ computations of the statement exactly right. ■

Actually, we have many more computations leading from P_0 to P_{n+1} under the assumptions of the above proposition. This depends on the decompositions that we can choose for the set $\{\theta_1, \ldots, \theta_n\}$. It is possible to consider sets with $2, \ldots, n-1$ elements and all their combinations such that the cardinality of the decompositions along with a computation from P_0 to P_{n+1} adds up to n. We can then

apply Theorem 4.1 generalized to higher dimension transitions to this computation. This discussion and Proposition 5.1 allow us to state the following theorem.

THEOREM 5.1. *Given a* $P_0 \xrightarrow{\{\theta_1,\ldots,\theta_n\}} P_{n+1}$ *transition in a higher dimension transition system, there are*

$$\sum_{h=1}^{n} \binom{n}{h} \sum_{i}^{n-h} \binom{n-h}{i}$$

computations leading from P_0 *to* P_{n+1} *whose transitions are labelled by subsets of* $\{\theta_1, \ldots, \theta_n\}$ *and for any computation* $P_0 \rightarrow^* P_{n+1}$, *its transitions are labelled by* $I_1, .., I_k$ $(k \leq n)$ *with* $\sum_{i=1}^{n} |I_i| = n$, $\cap_{i=1}^{k} I_i = \emptyset$, *and* $\cup_{i=1}^{k} I_i = \{\theta_1, \ldots, \theta_n\}$.

This theorem is a generalization of the classical result of the permutation of concurrent transitions. The number of computations in the theorem above is obtained by simple combinatorial reasoning. The empty intersection derives from the irref exivity of the concurrency relation.

We want to avoid all the computations that originate from generalized permutations of concurrent sets of transitions (see Def nition 5.1) to satisfy the maximal parallelism assumption.

DEFINITION 5.4. A *maximal parallelism transition system* is a higher dimension transition system *ts* generated according to the rule in Def nition 5.1 and such that

- $\forall P \xrightarrow{I} P' \in ts$, there is no $P_0, .., P_{n+1} \in ts$, $P_0 \xrightarrow{I_0} \cdots \xrightarrow{I_n} P_{n+1}$ with $P_0 = P$, $P_{n+1} = P'$, and $\bigcup_{i=0}^{n} I_i = I$.

A property of maximal parallelism transition systems is that their branching structure is only caused by nondeterministic choices. More formally, we have the following proposition.

PROPOSITION 5.2. *If* $P \xrightarrow{I_0} P_0$ *and* $P \xrightarrow{I_1} P_1$ *are two transitions of a maximal parallelism transition system, then* $I_0 \not\frown I_1$. *More precisely*,

$$\exists \theta_i \in I_0, \quad \theta_j \in I_1 \,.\, \theta_i \not\frown \theta_j.$$

Proof. By contradiction. Assume that $I_0 \smile I_1$. By a generalization of Theorem 4.1, there is P' such that $P_0 \xrightarrow{I_1} P'$ and $P_1 \xrightarrow{I_0} P'$. We can now apply the rule in Def nition 5.1 to derive the transition $P \xrightarrow{I_0 \cup I_1} P'$ against the condition stated in Def nition 5.4. ■

The maximal parallelism assumption allows one to simultaneously f re all the transitions enabled in a given state. Therefore, the notion of enabling is useless because the activity enabling the current transition is always f red by the transition immediately preceding it. More formally, we have the following proposition.

PROPOSITION 5.3. *If* $P_0 \xrightarrow{I_0} P_1 \xrightarrow{I_1} \cdots \xrightarrow{I_n} P_{n+1}$ *is a higher dimension computation, then, under the maximal parallelism assumption*,

$$\forall \theta \in I_i, \quad \exists \theta' \in I_{i-1} \,.\, \theta' \preceq \theta,$$

where $0 < i \leq n$. *In other words, the enabling relation between the transitions of a computation coincides with the temporal ordering of the transitions.*

Proof. By contradiction. Assume that $\forall \theta' \in I_{i-1} \,.\, \theta' \smile \theta$, $\theta \in I_i$. Then, according to the rule in Def nition 5.1, in the higher dimension transition under the maximal parallelism assumption, P_i' exists such that $P_i \xrightarrow{\{\theta\}} P_i' \xrightarrow{I_i'} P_{i+1}$ with $\{\theta\} \cup I_i' = I_i$. Furthermore, by applying the generalization to higher dimension transitions specif ed in Theorem 4.1 we can deduce that $P_{i-1} \xrightarrow{\{\theta\}} P' \xrightarrow{I_{i-1}} P_i'$ and $P_{i-1} \xrightarrow{I_{i-1}} P_i \xrightarrow{\{\theta\}} P_i'$. Now, Def nition 5.1 suff ces to derive the transition $P_{i-1} \xrightarrow{\{\theta\} \cup I_{i-1}} P_i'$, against the maximal parallelism assumption of the computation in the statement. ■

As a consequence of the above proposition, the sum $\sum_{h=i+1}^{n-1} \hat{T}_h$ in Definition 5.2 always reduces to zero. The distribution of the θ' activity is then $P(T' \leq t \mid T' > 0)$ and it turns out to be $\iota(\theta')$. This discussion allows us to state the following theorem.

THEOREM 5.2. *Under the maximal parallelism assumption, the enabling memory discipline reduces to* temporal ordering discipline; *i.e., the distribution in Definition* 5.2 *reduces to* $\iota(\theta')$.

The above theorem states that the enabling memory discipline is helpful only when the resources of the system are limited; that is, enabling is needed when concurrent transitions are sequentialized in some order due to a lack of resources (e.g., processors). An immediate corollary follows.

COROLLARY 5.1. *Any continuous distribution computed according to the enabling memory discipline enjoys the memoryless property under the maximal parallelism assumption.*

The above corollary holds only because the history of the system is condensed to the transition that immediately preceeds it. In other words, a transition is fired as soon as it becomes enabled.

6. PERFORMANCE BISIMULATION

In this section we extend the notion of late bisimulation of the π-calculus to cope with transition distributions. We call the new bisimulation *performance bisimulation*. We follow the approach introduced in [33] for CCS and arranged for PEPA in [30] and for TIPP in [28]. Since we relabelled the computations of our system with the distributions updated to consider both the branching structure and the history of the system, we simply need to compare the labels when checking bisimulation. This is the same idea adopted in [20] to define noninterleaving bisimulations for a qualitative analysis of systems. The uniform framework makes the integration of the analysis of qualitative and quantitative properties easy.

We must first recall the definition of parametric late bisimulation [20]. To determine the labels of transitions to be compared, we rely on a relabelling function that may inspect all the previous computation steps. Therefore our bisimulation will relate pairs $\langle P, \xi \rangle$, where P is the target state of the computation ξ.[3] We now adapt the definition of parametric bisimulation to performance transition systems.

DEFINITION 6.1. Given a relabelling function f, a binary relation $\mathcal{S}$ on pairs $\langle P, \xi \rangle$ of processes and performance computations is a *late f-simulation* if $\langle P, \xi \rangle \mathcal{S} \langle Q, \xi' \rangle$ implies that

- If $P \xrightarrow{\theta, F} P'$ and $\ell(\theta)$ is τ, $\overline{x}z$ or $\overline{x}(y)$ with $bn(\ell(\theta)) \notin fn(P, Q)$, then for some Q', $Q \xrightarrow{\theta', F'} Q'$, $f(\theta) = f(\theta')$, and $\langle P', \xi \xrightarrow{\theta, F} P' \rangle \mathcal{S} \langle Q', \xi' \xrightarrow{\theta', F'} Q' \rangle$;
- if $P \xrightarrow{\vartheta x(y), F} P'$ with $y \notin fn(P, Q)$, then for some Q', $Q \xrightarrow{\vartheta' x(y)} Q'$ and for all $w \in \mathcal{N}$, $\langle P'\{w/y\}, \xi \xrightarrow{\vartheta x(y), F} P'\{w/y\} \rangle \mathcal{S} \langle Q'\{w/y\}, \xi' \xrightarrow{\vartheta' x(y)} Q'\{w/y\} \rangle$.

The relation $\mathcal{S}$ is a *late f-bisimulation* if both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are late f-simulations. P is late f-bisimilar to Q (written $P \approx_f Q$) if there exists a late f-bisimulation $\mathcal{S}$ such that $\langle P, \epsilon \rangle \mathcal{S} \langle Q, \epsilon \rangle$.

If we put $f = \ell$ in the above definition, we recover the classical late bisimulation of the π-calculus [36].

The notion of f-bisimulation completely ignores the F component of the labels of performance transitions and thus it is only suitable to investigate the qualitative properties of processes. To cope with the quantitative information encoded in F, we raise the conditions in Definition 6.1 on transitions between states to conditions on transitions between equivalence classes of states. This is implemented via the function γ_0 in the following proposition that provides an alternative characterization of $\approx_f$ in the style of [28, 30, 33].

PROPOSITION 6.1. *Given a relabelling function f, a binary relation $\mathcal{S}$ on pairs $\langle P, \xi \rangle$ of processes and performance computations is a* late *f*-bisimulation *if $\langle P, \xi \rangle \mathcal{S} \langle Q, \xi' \rangle$ implies that for any equivalence*

[3] Although ξ uniquely determines P, we prefer to explicitly write out the process for readability. Also recall that the empty computation of P has P both as a source and as a target state.

class C which originated from $\mathcal{S}$

$$\forall\theta\,.\,\ell(\theta)\in\{\overline{x}y,\overline{x}(y),\tau,x(y)\}\,.\,\gamma_0(\langle P,\xi\rangle,\theta,C)=\gamma_0(\langle Q,\xi'\rangle,\theta',C),$$
$$f(\theta)=f(\theta')\ \textit{and}\ bn(\ell(\theta))\notin \mathrm{fn}(P,Q),$$

where

$$\gamma_0(\langle P,\xi\rangle,\theta,C)=\begin{cases}1 & \text{if } \exists\left\langle P',\xi\xrightarrow{\theta,F}P'\right\rangle\in C\,.\,\ell(\theta)\in\{\overline{x}y,\overline{x}(y),\tau\}\\ 1 & \textit{if } \forall w\in\mathcal{N},\exists\left\langle P'\{w/y\},\xi\xrightarrow{\theta,F}P'\{w/y\}\right\rangle\in C\\ & \qquad\qquad \ell(\theta)=x(y)\\ 0 & \textit{otherwise.}\end{cases}$$

Proof. Consider the case $\ell(\theta)=x(y)$. According to Def nition 6.1, $\langle P,\xi\rangle\,\mathcal{S}\,\langle Q,\xi'\rangle$ implies that whenever $P\xrightarrow{\theta,F}P'$, there exists Q' such that $Q\xrightarrow{\theta',F'}Q'$ with $f(\theta)=f(\theta')$ and $\forall w\in\mathcal{N}$, $\langle P'\{w/y\},\xi\xrightarrow{\theta,F}P'\{w/y\}\rangle\,\mathcal{S}\,\langle Q'\{w/y\},\xi'\xrightarrow{\theta',F'}Q'\{w/y\}\rangle$. Consequently, $\forall w\in\mathcal{N}$, $\langle Q'\{w/y\},\xi'\xrightarrow{\theta',F'}Q'\{w/y\}\rangle\in[\langle P'\{w/y\},\xi\xrightarrow{\theta,F}P'\{w/y\}\rangle]_{\mathcal{S}}$. By def nition of γ_0 we can conclude that

$$\gamma_0\Big(\langle P,\xi\rangle,\theta,\Big[\Big\langle P'\{w/y\},\xi\xrightarrow{\theta,F}P'\{w/y\}\Big\rangle\Big]_{\mathcal{S}}\Big)=1$$
$$=\gamma_0\Big(\langle Q,\xi'\rangle,\theta',\Big[\Big\langle P'\{w/y\},\xi\xrightarrow{\theta,F}P'\{w/y\}\Big\rangle\Big]_{\mathcal{S}}\Big).$$

Since we imposed $f(\theta)=f(\theta')$, we proved that $\mathcal{S}$ is a late f-simulation. Repeating the above starting with $Q\xrightarrow{\theta',F'}Q'$, we can conclude that $\mathcal{S}^{-1}$ is a late f-simulation as well and hence that $\mathcal{S}$ is a late f-bisimulation.

The cases $\ell(\theta)\in\{\overline{x}y,\overline{x}(y),\tau\}$ are simpler and similar. ■

We instantiate the late f-bisimulation with our structural enabling relation $\preceq_{str}$.[4] The motivation is twofold. First, since we cope with general distributions, we feel it is natural to include in the def nition of bisimulation the same notion of dependency used to correctly compute the distributions of performance transitions. Second, structural enabling allows us to state a congruence property for our equivalence relation that is missing for classical π-calculus late bisimulation (see [5]). We can now def ne an enabling relabelling function to instantiate the f function in Def nition 6.1.

Definition 6.2. Given a performance computation $\xi=P_0\xrightarrow{\theta_0,F_0}P_1\xrightarrow{\theta_1,F_1}\cdots\xrightarrow{\theta_n,F_n}P_n$, its associated enabling computation $Et(\xi)$ is derived by relabelling any transition θ_k, F_k as et_k, where

$$et_k=\begin{cases}\tau,F_k & \text{if } \ell(\theta_k)=\tau\\ \langle\ell(\theta_k),\{h\neq k|\theta_h\preceq_{str}\theta_k,\ell(\theta_h)\neq\tau\}\rangle,F_k & \text{otherwise.}\end{cases}$$

By abuse of notation we will sometimes write $Et(\theta_k)$ in place of et_k.

To introduce our performance bisimulation we also need to modify the def nition of the function γ_0 in Proposition 6 in order to take the duration of transitions into account. (Remember that we write T_i for the random variable, which describes the time interval of a $P\xrightarrow{\theta_i,F_i}P_i$ transition.)

Definition 6.3. A binary relation $\mathcal{S}$ on pairs $\langle P,\xi\rangle$ of processes and performance computations is a *performance bisimulation* if $\langle P,\xi\rangle\,\mathcal{S}\,\langle Q,\xi'\rangle$ implies that for any equivalence class C originating

[4] We can use the full enabling relation $\preceq$ without altering the following results. We prefer to rely on $\preceq_{str}$ to simplify the technical presentation.

from $\mathcal{S}$

$$\forall\theta\,.\,\ell(\theta)\in\{\overline{x}y,\overline{x}(y),\tau,x(y)\}\,.\,\gamma(\langle P,\xi\rangle,\theta,C)=\gamma(\langle Q,\xi'\rangle,\theta',C),$$
$$Et(\theta)=Et(\theta')\text{ and }bn(\ell(\theta))\notin fn(P,Q),$$

where

$$\gamma(\langle P,\xi\rangle,\theta,C)=\begin{cases}\mathcal{P}r\Big(T_h\le t\mid T_h=\min\{T_i\},\exists\Big\langle P',\xi\xrightarrow{\theta,F}P'\Big\rangle\in C\,.P\xrightarrow{\langle\theta,F\rangle}P'\Big) & \text{if }\ell(\theta)\neq x(y)\\ \mathcal{P}r\Big(T_h\le t\mid T_h=\min\{T_i\},\forall w\in\mathcal{N},\quad\exists\Big\langle P'\{w/y\},\xi\xrightarrow{\theta,F}P'\{w/y\}\Big\rangle\in C\,.\\ \quad P\xrightarrow{\langle\theta,F\rangle}P'\{w/y\}\Big)\text{ otherwise.}\end{cases}$$

P is performance bisimilar to Q (written $P\approx_P Q$) if there is a performance bisimulation $\mathcal{S}$ such that $\langle P,\epsilon\rangle\,\mathcal{S}\,\langle Q,\epsilon\rangle$.

In the above definition we consider the maximum transition speed from one equivalence class to another via the *same* action label μ (indeed $Et(\theta)=Et(\theta')$ implies $\ell(\theta)=\ell(\theta')$). In other words, we compare two systems on their fastest runs. Hence, the usage of the min operator in the definition of γ.

An important issue in the definition of equivalences in the setting of stochastic process algebras concerns the compositional minimization of system descriptions. We prove a congruence result for our performance bisimulation below, following the same pattern used in [5] for their causal bisimulation. We start with some auxiliary definitions and lemmas.

DEFINITION 6.4. A substitution $\sigma:\mathcal{N}\to\mathcal{N}$ is $\$_a$-preserving iff $\sigma(x)=y$ implies that $\$_a(x)=\$_a(y)$.

Hereafter we write $\theta\sigma$, meaning σ applied to $\ell(\theta)$, and assume any substitution to be $\$_a$-preserving. The following lemma shows how substitutions affect transitions.

LEMMA 6.1. *If σ is a $\$_a$-preserving substitution and P a process, then*

1. $P\xrightarrow{\theta,F}P'\;\Rightarrow\;P\sigma\xrightarrow{\theta\sigma,F}P'\sigma,\ell(\theta)\neq\tau,Et(\theta)=Et(\theta\sigma)$;
2. $P\xrightarrow{\vartheta\tau,F}P'\;\Rightarrow\;P\sigma\xrightarrow{\vartheta\tau,F}P'\sigma$;
3. $P\xrightarrow{\vartheta\langle\theta_0,\theta_1\rangle,F}P'\;\Rightarrow\;P\sigma\xrightarrow{\vartheta\langle\theta_0\sigma,\theta_1\sigma\rangle,F}P'\sigma,Et(\vartheta\theta_0)=Et(\vartheta\theta_0\sigma),Et(\vartheta\theta_1)=Et(\vartheta\theta_1\sigma)$;

Proof. The proof is clear by noting that σ only affects the action names that are not considered at all by the definition of $\preceq_{str}$ (see Definition 4.1) and by applying the same result for standard late bisimulation [36]. ■

We now use a simple proof technique developed in [36] and based on ground bisimilarity.

DEFINITION 6.5. A binary relation $\mathcal{S}$ on pairs $\langle P,\xi\rangle$ of processes and performance computations is a *ground performance bisimulation up to $\equiv$ and restriction* if $\langle P,\xi\rangle\,\mathcal{S}\,\langle Q,\xi'\rangle$ implies that for any equivalence class C originating from $\mathcal{S}$

$$\gamma'(\langle P,\xi\rangle,\theta,C)=\gamma'(\langle Q,\xi'\rangle,\theta',C),\ Et(\theta)=Et(\theta')\text{ and }bn(\ell(\theta))\notin fn(P,Q),$$

where

$$\gamma'(\langle P,\xi\rangle,\theta,C)=\mathcal{P}r\Big(T_h\le t\mid T_h=\min\{T_i\},\exists\Big\langle P',\xi\xrightarrow{\theta,F}P'\Big\rangle\in C\,.\,P\xrightarrow{\langle\theta,F\rangle}P'\Big)$$

and $\langle P',\xi\xrightarrow{\theta,F}P'\rangle,\langle Q',\xi\xrightarrow{\theta',F'}Q'\rangle\in C$ and $P'\equiv(\nu\,\tilde{b})P_1$, $Q'\equiv(\nu\,\tilde{b})Q_1$.

P is ground performance bisimilar to Q (written $P\approx_P^g Q$) if there is a ground performance bisimulation $\mathcal{S}$ such that $\langle P,\epsilon\rangle\,\mathcal{S}\,\langle Q,\epsilon\rangle$.

Standard concurrency theory [36] establishes the relation between ground and nonground bisimulations for the π-calculus.

LEMMA 6.2. *If $\mathcal{S}$ is a ground performance bisimulation up to $\equiv$ and restriction, then $\mathcal{S} \subseteq \approx_P$.*

Theorem 6.1 states that ground performance bisimulation is preserved by substitutions.

THEOREM 6.1. *If σ is a $\$_a$-preserving substitution, then $\langle P, \xi \rangle \approx_P^g \langle Q, \xi' \rangle$ implies $\langle P\sigma, \xi\sigma \rangle \approx_P^g \langle Q\sigma, \xi'\sigma \rangle$.*

Proof. Mimicking [5], we show that

$$\mathcal{S} = \{(\langle P\sigma, \xi\sigma \rangle, \langle Q\sigma, \xi'\sigma \rangle) | \langle P, \xi \rangle \approx_P^g \langle Q, \xi' \rangle \ \wedge \ \sigma \text{ is } \$_a\text{-preserving}\}$$

is a ground performance bisimulation up to $\equiv$ and restriction. For any $P\sigma \xrightarrow{\theta, F} P'$ we have to find P_1, Q_1, $\tilde{b}$, σ' such that $Q\sigma \xrightarrow{\theta', F'} Q'$ with $P' \equiv (\nu\, \tilde{b})P_1\sigma'$, $Q' \equiv (\nu\, \tilde{b})Q_1\sigma'$, $Et(\theta) = Et(\theta')$, and $\gamma'(\langle P\sigma, \xi\sigma \rangle, \theta, C) = \gamma'(\langle Q\sigma, \xi'\sigma \rangle, \theta', C)$. Theorem 4.11 in [5] and Lemma 6.1 guarantee the first two conditions of the statement above. We only have to check the equality of the two γ'-functions. Since $\$_a$-preserving substitutions do not affect the duration of transitions (Lemma 6.1), and since $\langle P, \xi \rangle \approx_P^g \langle Q, \xi' \rangle$ implies $\gamma'(\langle P, \xi \rangle, \theta, C) = \gamma'(\langle Q, \xi' \rangle, \theta', C)$, we have proven the theorem. ■

Since to obtain congruences one has to expect that bisimilarity is preserved by substitutions, a corollary of Theorem 6.1 states a congruence result for $\approx_P^g$.

COROLLARY 6.1. *$\approx_P^g$ is a congruence for the π-calculus without matching.*

Eventually we prove that our performance bisimulation is indeed a congruence.

THEOREM 6.2. *$\approx_P$ is a congruence for the π-calculus without matching.*

Proof. We only need to show that $\approx_P^g$ and $\approx_P$ coincide on the considered subset of the π-calculus. To prove the inclusion $\approx_P \subseteq \approx_P^g$, we only need to note that all the requisites in the definition of $\approx_P^g$ also occur in the definition of $\approx_P$. To prove the converse inclusion we show that $\approx_P^g$ is a performance bisimulation. The only point that needs to be checked is the input clause in Proposition 6.1, which is satisfied because $\approx_P^g$ is closed under $\$_a$-preserving substitutions (Theorem 6.1). ■

7. EXPONENTIAL DISTRIBUTIONS

In this section we show how our definitions and results change when only exponential distributions are around. This is actually the only case considered in most papers on Markovian process algebras. We start by remembering what exponential distributions are.

An exponential distribution with rate r is a function $F(t) = 1 - e^{-rt}$, where t is the time parameter. The parameter r determines the shape of the curve. The greater the r parameter, the faster $F(t)$ approaches its asymptotic value. The probability of performing an action with parameter r within time t is $F(t) = 1 - e^{-rt}$, so r determines the time Δt needed to obtain a probability near to 1. The exponential density function is $f(t) = re^{-rt}$.

Exponential distributions have the *memoryless property*. Roughly speaking, transitions occur independent of when the last transition occurred. In other words, how long the transition waits before completion does not depend on how long it has already waited. Thus, the time elapsed by an activity in a state where another one is the fastest is useless. This means that any time a transition becomes enabled, it restarts its elapsing time just as it would the first time it is enabled. Consequently, the treatment of enabling memory discipline has no counterpart in a pure exponential setting.

Let us discuss how the definition of the cost function $\$ changes. First, we say $\$_a(\mu) = \lambda \in \mathbb{R}^+$, where λ is the single parameter uniquely describing an exponential distribution. Then, $\$_o(\vartheta) = r \in (0..1]$. We can define $\$_o$ as in Section 3 by simply exchanging $\sum_{o \in \vartheta}$ with $\prod_{o \in \vartheta}$ to ensure that $\$_o(\vartheta) \leq 1$. Since we will use r as a multiplicative factor for λ in the definition of $\$, the interval (0..1] is the domain of a slowing-down parameter. Similar to $\$_o$, we have $f_{\langle\rangle} : \mathcal{L}^* \times \mathcal{L}^* \to (0..1]$. Finally, Definition 3.1 becomes

DEFINITION 7.1. *The function* $\$: \Theta \to \mathbb{R}^+$ *is defined as*

$$\$(\vartheta\,\mu) = \$_o(\vartheta) \times \$_a(\mu)$$
$$\$(\vartheta\,\langle \vartheta_0\alpha_0, \vartheta_1\alpha_1\rangle) = f_{\langle\rangle}(\mathfrak{w}_0, \mathfrak{w}_1) \times \min\{\$(\vartheta\,\vartheta_0\alpha_0), \$(\vartheta\,\vartheta_1\alpha_1)\}.$$

Our knowledge on the kind of distributions we are dealing with enables us to simplify the expression of Theorem 3.1. The proof of the following theorem only amounts to computing the integrals in Theorem 3.1 by letting $F_i(t)$ be $1 - e^{-\lambda_i t}$.

THEOREM 7.1. *Given a process* P,

$$R_P = \sum_{P \xrightarrow{\theta_i,\lambda_i} P_i} \lambda_i$$

is the exit rate *of* P. *Then, the probability of* $P \xrightarrow{\theta_i,\lambda_i} P_i$ *is*

$$\frac{\lambda_i}{R_P};$$

the distribution of the random variable T_i *which describes the time interval associated with* $P \xrightarrow{\theta_i,\lambda_i} P_i$ *is*

$$\tilde{F}_i(t) = 1 - e^{-R_P t};$$

the apparent rate of an action a *in* P *is*

$$r_a(P) = \frac{1}{R_P} \sum_{\substack{P \xrightarrow{\theta_j,\lambda_j} P_j \\ \ell(\theta_j)=a}} \lambda_j;$$

the probability of $P \xrightarrow{\theta_i,\lambda_i} P_i$, $\ell(\theta_i) = a$, *given that an action* a *occurred is*

$$\frac{\lambda_i}{\sum_{\substack{P \xrightarrow{\theta_j,\lambda_j} P_j \\ \ell(\theta_j)=a}} \lambda_j}.$$

We can now instantiate performance bisimulation to exponential distributions. In spite of the interleaving nature of exponential distributions, we still use the structural enabling relation in order to preserve the congruence result of our equivalence. We only need to work on function γ in Def nition 6.3 by putting any distribution $F_i = 1 - e^{r_i t}$. To make understanding it easier, we denote the resulting function $\hat{\gamma}$.

DEFINITION 7.2. The *exponential performance bisimulation* (written $\approx^e_P$) is def ned as in Def nition 6.3 except that function γ is replaced by $\hat{\gamma}$, def ned by interpreting $\langle P_i, \xi_i\rangle$ as a singleton equivalence class as

$$\hat{\gamma}(\langle P, \xi\rangle, \theta, C) = \sum_{\langle P_i,\xi_i\rangle \in C, \ell(\theta_i)=\ell(\theta)} \hat{\gamma}(\langle P, \xi\rangle, \theta, \langle P_i, \xi_i\rangle) = \sum_{\langle P_i,\xi_i\rangle \in C, \ell(\theta_i)=\ell(\theta)} r_i,$$

where r_i is the exponential distribution associated with a $P \xrightarrow{\theta_i,r_i} P_i$ transition where $\ell(\theta_i) = \ell(\theta)$.

Remember that the condition $Et(\theta) = Et(\theta')$ in the above def nition ensures that $\hat{\gamma}$ def nes the *total conditional transition rate* as def ned in [30] for PEPA.

We can now show how the exponential performance bisimulation induces a lumpable partition on the underlying Markov process. First we need some auxiliary definitions and lemmas.

DEFINITION 7.3. Given a Markov process with state space $\{X_1, X_2, \ldots, X_n\}$, the state space of the *aggregated process* is some partition $\rho = \{X_{[1]}, X_{[2]}, \ldots, X_{[n]}\}$ with $N < n$ obtained through an equivalence relation on the states of the original process.

A Markov process is *lumpable* compared to a ρ partition if for every initial distribution the aggregated process is a Markov process.

The following lemma establishes that two exponential performance bisimilar processes have the same total unconditional transition rate of entering a C equivalence class.

LEMMA 7.1. *If* $\langle P, \xi\rangle \approx^e_P \langle Q, \xi'\rangle$ *and* $p(\langle P, \xi\rangle, C)$ *denotes the probability of reaching step a state* $P' \in C$ *from* P *in just* 1, *then*

$$\frac{\sum_{\ell(\theta_i)\in\mathcal{N}} \hat{\gamma}(\langle P, \xi\rangle, \theta_i, C)}{\sum_{P \xrightarrow{\theta_i, r_i} P_i} r_i} = \frac{\sum_{\ell(\theta'_i)\in\mathcal{N}} \hat{\gamma}(\langle Q, \xi'\rangle, \theta'_i, C)}{\sum_{Q \xrightarrow{\theta_i, r_i} Q_i} r_i}.$$

We eventually prove the lumpability result by defining the state space of a P *process as* $ds(\langle P, \epsilon\rangle) = \{\langle P_i, \xi_i\rangle \mid \exists \xi_i = P \xrightarrow{\theta_1, r_1} P_1 \xrightarrow{\theta_2, r_2} \cdots \xrightarrow{\theta_i, r_i} P_i\}$.

THEOREM 7.2. *For any* P *process,* $ds(\langle P, \epsilon\rangle)_{/\approx^e_P}$ *induces a lumpable partition on the state space of the Markov process which corresponds to* P.

Proof. Let $C_i, C_j \in ds(\langle P, \epsilon\rangle)_{/\approx^e_P}$ and consider $P'_i, P''_i \in C_i$. According to Lemma 7.1, it is $\sum_{\ell(\theta_i)\in\mathcal{N}} \hat{\gamma}(\langle P'_i, \xi\rangle, \theta_i, C_j) = \sum_{\ell(\theta'_i)\in\mathcal{N}} \hat{\gamma}(\langle P''_i, \xi'\rangle, \theta'_i, C_j)$ because $P'_i \approx^e_P P''_i$. Since [32] proved that a Markov process is lumpable compared to a $\rho = \{X_{[i]}\}$ partition if and only if for any $X_{[k]}, X_{[l]} \in \rho$, $X_i, X_j \in X_{[k]}$ the total transition rate from X_i and X_j to $X_{[l]}$ is the same, we have proven the theorem. ■

We now outline how the distribution of synchronizations can be specified in our approach which is similar to that of the PEPA language. We first recall that a PEPA-like synchronization rule establishes that the rate of a τ-transition between a $\langle a, r_0\rangle$ prefix of a process P and a $\langle \overline{a}, r_1\rangle$ prefix of a process Q is

$$r' = \frac{r_0}{R_a(P)} \times \frac{r_1}{R_a(Q)} \times \min\{R_a(P), R_a(Q)\}.$$

The corresponding transition in the proved semantics is labelled $\vartheta\langle ||_0\vartheta_0 a, ||_1\vartheta_1\overline{a}\rangle$. To recover the rate r' we only need to define

$$f_{\langle\rangle}(\mathfrak{A}||_0\mathfrak{A}_0, \mathfrak{A}||_1\mathfrak{A}_1) = \frac{\min\{R_a(P), R_a(Q)\}}{R_a(P) \times R_a(Q)}$$

with $\bullet$ of Definition 3.1 being the multiplication. Recall that $\mathfrak{A}||_0\mathfrak{A}_0$ and $\mathfrak{A}||_1\mathfrak{A}_1$ uniquely identify the sequential components that figure the complementary transitions. Therefore we can compute $R_a(P)$ and $R_a(Q)$) according to Theorem 7.1. Then we replace $\min\{\$(\vartheta\vartheta_0\alpha_0), \$(\vartheta\vartheta_1\alpha_1)\}$ from Definition 7.1 with $\$(\vartheta||_0\vartheta_0 a) \times \$(\vartheta||_1\vartheta_1\overline{a})$ and define $\$(\vartheta||_0\vartheta_0 a) = r_0$ and $\$(\vartheta||_1\vartheta_1\overline{a}) = r_1$. Note that $\vartheta||_0\vartheta_0 a$ and $\vartheta||_1\vartheta_1\overline{a}$ uniquely identify the prefix corresponding to the transition. Thus we can define $ in a tabular way because the prefixes of a program are finite.

We end this section by referring any reader interested in exponential distributions to [10, 40, 41] for case studies based on our framework.

8. RELATED WORK

The earliest study of stochastic process algebras with general distributions can be found in [25], which adds structure to the states of *TIPP* transition systems. The operational semantics uses counters to keep track of how many times an action occurring in a parallel composition has not been selected to happen.

Similar semantics are studied in [9, 31] through a stochastic extension of bundle event structures with phase-type distributions. This model copes more naturally with general distributions because activities that are not causally dependent are not related in the model (as in our approach) contrary to what happens in interleaving models. The authors present a process algebra as well whose semantics is def ned in terms of the stochastic event structures. There is also a stochastic extension to *LOTOS* where distributions are not limited to exponential ones [2], provided that all the synchronizations are known at the top level.

A process algebra for discrete event simulation is introduced in [27]. A timer is set randomly and its expiration determines the actions to be f red. Since urgent and delayable pref xes are considered, it is possible to cope with both open and closed systems. A similar approach is presented in [15, 16] for the process algebra Spades where the f niteness of semantic objects is maintained. There is an expansion law for decomposing parallel compositions of Spades processes.

A general semi-Markovian process algebra is obtained by working on EMPA in [7, 8], which interprets processes in ST-semantics which allows for the ref nement of actions. This process algebra represents the GSMP model in a complete way.

The main differences between the approaches above and ours are that they do not cope naturally with mobility and they need to incorporate the stochastic information into the calculus syntax.

Finally, note that the literature presents other attempts at including quantitative information in process algebras for performance evaluation proposed. There are two approaches: the probabilistic and the temporal. *Probabilistic process algebras* rule out nondeterminism by attaching probabilities to branching points. For instance see [26, 33, 53], but almost all proposals deal with synchronous calculi, thus limiting expressiveness. *Temporal process algebras* (for a survey see [38]) use time information to evaluate the duration of a specif c execution either by associating f xed durations to all actions with the same name or by interleaving explicit timed steps with action steps. The absolute duration of actions is sometimes unreal because the time needed by an action heavily depends on the state of resources, the conf icts for accessing them, and so on. In any case, the duration of a specif c execution does not provide the grounds for a performance evaluation of the whole system.

9. CONCLUSIONS

We have proposed a general framework for handling performance and behavioral analysis within the same model. The uniqueness of the model is a necessary condition to implement our ideas within compiler tools. Our approach is distinctly different from stochastic process algebras because it does not require the designer to know architectural constraints at the time of specif cation. In fact, no probabilistic information is inserted in the syntax of the language. In spite of this, we can derive the quantitative information we need by looking at the derivation trees of transitions encoded in their labels. The intuitive idea is that the semantic rules correspond to low-level routines of the run-time support of the language. We applied this framework in [40, 41] to model a network management system which yields the same performance results reported in [3] and obtained through informal reasoning. We also studied a distributed database application and recovered the same quantitative parameter obtained in [13] by informal reasoning. These analyses were carried out by using a tool which implements our framework in an exponential setting [10].

We use the information carried over by our enhanced labels to handle general continuous distributions as well. In particular, we resort to the notion of enabling between transitions [20] to take the history of the system into account. The notion of enabling naturally leads to the enabling memory discipline for selecting which transitions to f re [1]. We prove that this notion works properly with limited resources. Under a maximal parallelism assumption (unbound resources), we ref ned the enabling memory discipline so that the history of the system is no longer needed even for general continuous distributions. The main reason is that a transition is either f red as soon as it becomes enabled or it is discarded.

Our current studies concern ref nement of the def nition of function $ to obtain sensible transition costs and to characterize its algebraic properties. We also plan to extend this framework to real distributed languages and to test how it is scalable on real applications. Furthermore, we implemented a stochastic version of the full π-calculus to simulate the behavior of complex systems [48, 49].

As a further investigation, it would be interesting to see how generalized semi-Markov processes, which are receiving great attention in the literature relate to our approach.

This paper is a further step towards def ning a kernel of an integrated environment for creating distributed and mobile systems. To this purpose, we feel it is essential not to modify the syntax of the language in order to retrieve or restore the information needed to carry out both qualitative and quantitative analyses. This amounts to saying that the designer of a system only needs to know the syntax of the (specif cation) language. Implementation-dependent information is encoded in the compiler. Furthermore, the topic of this paper further stresses the motto *proof as transitions* described in [19] where the authors claim that enhanced labels encode almost all the information needed for software development.

ACKNOWLEDGMENTS

The author thanks Giandomenico Orlandi for his suggestions on the mathematical treatment of the homotety of arguments. The author also thanks Paola Siri and Stefania Ugolini for their help on probabilistic matters. Finally, the author is grateful to the reviewers for their precise and valuable comments. The author is supported by the DEGAS 1ST-2001-32072 under the FET proactive initiative on global computing.

REFERENCES

1. Ajmone Marsan, M., Balbo, G., Bobbio, A., Chiola, G., Conte, G., and Cumani, A. (1989), The effect of execution policies on the semantics and analysis of stochastic Petri nets, *IEEE Trans. Software Eng.* **15**, 832–846.
2. Ajmone Marsan, M., Bianco, A., Ciminiera, L., Sisto, R., and Valenzano, A (1994), A LOTOS extension for the performance analysis of distributed systems, *IEEE/ACM Trans. Networking* **2**, 151–164.
3. Baldi, M., and Picco, G. P. (1998), Evaluating the tradeoffs of mobile code design paradigms in network management applications, *in* "Proceedings of ICSE'98," Assoc. Comput. Mach., New York.
4. Bernardo, M., Donatiello L., and Gorrieri, R. (1998), A formal approach to the integration of performance aspects in the modelling and analysis of concurrent systems, *Inform. and Comput.* **144**, 83–154.
5. Boreale, M., and Sangiorgi, D. (1998), Some congruence properties for π-calculus bisimilarities, *Theoret. Comput. Sci.* **198**.
6. Boudol, G., and Castellani, I. (1988), A non-interleaving semantics for CCS based on proved transitions, *Fund. Inform.* **XI**, 433–452.
7. Bravetti, M., Bernardo, M., and Gorrieri, R. (1998), Towards performance evaluation with general distributions in process algebra, *in* "Proceedings of CONCUR'98", Lecture Notes in Computer Science, Vol. 1466, pp. 405–422, Springer-Verlag.Berlin/New York.
8. Bravetti, M., and Gorrieri, R. (1999), Interactive generalized semi-Markov processes, *in* "Proceedings of PAPM'99."
9. Brinksma, E., Katoen, J.-P., Langerak, R., and Latella, D. (1995), A stochastic causality based process algebra, *Comput. J.* **38**, 552–565.
10. Brodo, L., Degano, P., and Priami, C. (2000), A tool for quantitative analysis of π-calculus processes, *in* "Proceedings of PAPM'00," Carleton Scientif c, Geneva.
11. Buchholz, P. (1994), "On a Markovian Process Algebra," Technical Report 500, Informatik IV, University of Dortmund.
12. Cardelli, L., and Gordon, A. (1998), Mobile ambients, *in* "Proceedings of FoSSaCS'98," Lecture Notes in Computer Science, Vol. 1378, pp. 140–155, Springer-Verlag, Berlin.
13. Carzaniga, A., Picco, G. P., and Vigna, G. (1997), Designing distributed applications with mobile code paradigms, *in* "Proceedings of ICSE'97," pp. 23–32, Assoc. Comput. Mach., New York.
14. Cleaveland, R. (1996), Position statement of the concurrency working group, *in* "Proceedings of ACM Workshop on Strategic Directions in Computing Research," Assoc. Comput. Mach., New York.
15. D'Argenio, P. R. (1999), "Algebras and Automata for Timed and Stochastic Systems," Ph.D. thesis, University of Twente.
16. D'Argenio, P. R., Katoen, J.-P., and Brinksma, E. (1998), An algebraic approach to the specif cation of stochastic systems, *in* "Proceedings of PROCOMET'98," IFIP Series, pp. 126–147, Chapman & Hall, London/New York.
17. Degano, P., De Nicola, R., and Montanari, U. (1985), Partial ordering derivations for CCS, *in* "Proceedings of FCT'85," Lecture Notes Computer Science, Vol. 199. pp. 520–533, Springer-Verlag, Berlin.
18. Degano, P., and Priami, C. (1992), Proved trees, *in* "Proceedings of ICALP'92," Lecture Notes in Computer Science, Vol. 623, pp. 629–640, Springer-Verlag, Berlin.
19. Degano, P., and Priami, C. (1996), Enhanced operational semantics, *ACM Comput. Surveys* **28**, 352–354.
20. Degano, P., and Priami, C. (1999), Non interleaving semantics for mobile processes, *Theoret. Comput. Sci.* **216**, 237–270.
21. Degano, P., Priami, C., Leth, L., and Thomsen, B. (1999), Causality for debugging mobile agents, *Acta Inform*. **36**, 335–374.
22. Fournet, C., Gonthier, G., Levy, J.-J. Maranget, L., and Remy, D. (1996), A calculus of mobile agents, *in* "Proceedings of CONCUR'96," Lecture Notes in Computer Science, Vol. 1119, pp. 406–421, Springer-Verlag, Berlin.
23. Giacalone, A., Mishra, P., and Prasad, S. (1989), Facile: A symmetric integration of concurrent and functional programming, *Internat. J. Parallel Programming* **18**, 121–160.
24. Götz, N., Herzog, U., and Rettelbach, M. (1992), "TIPP—A Language for Timed Processes and Performance Evaluation," Technical Report 4/92, IMMD VII, University of Erlangen-Nurnberg.

25. Götz, N., Herzog, U., and Rettelbach, M. (1993), TIPP—Introduction and application to protocol performance analysis, *in* "Formale Beshreibungstechniken für verteilte Systeme," FOKUS series, Saur Publishers, Munich.
26. Hansson, H., and Jonsson, B. (1990), A calculus of communicating systems with time and probability, *in* "Proceedings of IEEE RTSS'90, Orlando, Florida."
27. Harrison, P. G., and Strulo, B. (1995), Stochastic process algebra for discrete event simulation, "Quantitative Methods in Parallel Systems," pp. 18–37.
28. Hermanns, H., Herzog, U., and Mertsiotakis, V. (1998), Stochastic process algebras—between LOTOS and Markov chains, *Comput. Networks and ISDN System* **30**, 901–924,
29. Herzog, U. (1996), "A concept for graph-based process algebras, generally distributed activity times and hierarchical modelling, *in* "Proceedings of PAPM'96," pp. 1–20. CLUT, Torino.
30. Hillston, J. (1996), "A Compositional Approach to Performance Modelling," Cambridge Univ. Press, Cambridge, UK.
31. Katoen, J.-P., (1996), "Quantitative and Qualitative Extensions of Event Structures," Ph.D. thesis, Center for Telematics and Information Technology, University of Twente. Available as CTIT No. 96-09.
32. Kemeny, J. G., and Snell, J. L. (1960), "Finite Markov Chains," Van Nostrand, Princeton, NJ.
33. Larsen, K. G., and Skou, A. (1992), Compositional verif cation of probabilistic processes, *in* "Proceedings of CONCUR'92," Lecture Notes in Computer Science, Vol. 630, Springer-Verlag, Berlin.
34. Maggiolo-Schettini, A., and Tini, S. (1999), Applying techniques of asynchronous concurrency to synchronous languages, *Fund. Inform.* **40**, 221–250.
35. Milner, R. (1989), "Communication and Concurrency," Prentice-Hall, London.
36. Milner, R., Parrow, J., and Walker, D. (1992), A calculus of mobile processes, I, II, *Inform. and Comput.* **100**, 1–77.
37. Milner, R., Parrow, J., and Walker, D. (1993), Modal logics for mobile processes, *Theoret. Comput. Sci.* **114**, 149–171.
38. Nicollin, X., and Sifakis, J. (1991), An overview and synthesis on timed process algebras, "Real Time: Theory in Practice," Lecture Notes in Computer Science, Vol. 600, 526–548, Springer-Verlag, Berlin.
39. Nielson, F., and Nielson, H. R. (1996), From CML to its process algebra, *Theoret. Comput. Sci.* **155**, 179–219.
40. Nottegar, C., Priami, C., and Degano, P. (2001), Performance evaluation of mobile processes via abstract machines, *IEEE TSE* **27**, 867–889.
41. Nottegar, C., Priami, C., and Degano. P. (1999), Semantic-driven performance evaluation, *in* "Proceedings of FASE'99," Lecture Notes in Computer Science, Vol. 1577, pp. 204–218, Springer-Verlag, Berlin.
42. Pierce, B. C., and Turner, D. N. (1999), PICT: A programming language based on the pi-calculus, *in* "Proof, Language and Interaction: Essays in Honour of Robin Milner," MIT Press, Cambridge, MA.
43. Plotkin, G. (1981), "A Structural Approach to Operational Semantics," Technical Report DAIMI FN-19, Aarhus University, Denmark.
44. Priami, C. (1995), Stochastic π-calculus. *Comput. J.* **38**, 578–589.
45. Priami, C. (1996), Integrating behavioural and performance analysis with topology information, *in* "Proceedings of 29th Hawaian International Conference on System Sciences, Maui, Hawaii," Vol. 1, pp. 508–516, IEEE Press, New York.
46. Priami, C. (1996), Stochastic π-calculus with general distributions, *in* "Proceedings of PAPM'96," pp. 41–57, CLUT, Torino.
47. Priami, C. (1998), Enabling and general distributions in stochastic process algebras, *in* "Proceedings of Italian Conference on Theoretical Computer Science, Prato," pp. 192–203, World Scientif c, Singapore.
48. Priami, C., Regev, A., Shapiro, U., and Silverman, W. (2001), Application of a stochastic name-parsing calculus to representation and simulation of molecular processes, *Inf. Process. Lett.* **80**, 25–31.
49. Regev, A., Priami, C., Silverman, W., and Shapiro, U. Stochastic process algebras for the modeling of biomolecular processes. submitted for publication.
50. Riely, J., and Hennessy, M. (1998), A typed language for distributed mobile processes, *in* "Proceedings of POPL'98," pp. 378–390.
51. Sangiorgi, D. (1992), "Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms," Ph.D. thesis, University of Edinburgh.
52. Thomsen, B. (1993), Plain CHOCS: a second generation calculus for higher order processes, *Acta Inform*. **30**, 1–59.
53. van Glabbeek, R. J., Smolka, S. A., and Steffen, B. (1995), Reactive, generative and stratif ed models of probabilistic processes, *Inform. and Comput*. **121**, 59–80.